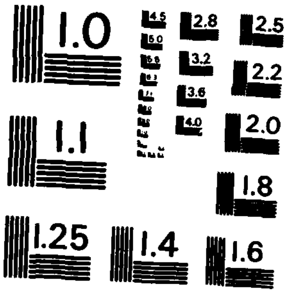
AD-A144 125 COMPREHENSIVE OCCUPATIONAL DATA ANALYSIS PROGRAMS 80 1/2
(CODAP80) USER'S MANUAL(U) NAVY OCCUPATIONAL
DEVELOPMENT AND ANALYSIS CENTER WASHINGTON DC JAN 84
UNCLASSIFIED DOD/DF-84/006A F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

50272-101

REPORT DOCUMENTATION PAGE

1. REPORT NO. DOD/DF-84/006a

2. BIN-601

3. Recipient's Accession No.

4. Title and Subtitle
COMPREHENSIVE OCCUPATIONAL DATA ANALYSIS PROGRAMS 80 (CODAP80)
User's Manual

5. Report Date
JANUARY 1984

6.

7. Author(s)
N/A

8. Performing Organization Rept. No.

9. Performing Organization Name and Address
NAVY OCCUPATIONAL DEVELOPMENT AND ANALYSIS CENTER (NODAC)
BUILDING 150, WASHINGTON NAVY YARD (ANACOSTIA)
WASHINGTON, DC 20374

10. Project/Task/Work Unit No.

11. Contract(C) or Grant(G) No.
(C) N/A
(G) N/A

12. Sponsoring Organization Name and Address
NAVY OCCUPATIONAL DEVELOPMENT AND ANALYSIS CENTER (NODAC)
BUILDING 150, WASHINGTON NAVY YARD (ANACOSTIA)
WASHINGTON, DC 20374

13. Type of Report & Period Covered
FINAL RELEASE 83.1

14.

15. Supplementary Notes

SOURCE CODE FOR CODAP80 PROGRAMS.
for magnetic tape see

16. Abstract (Limit: 200 words)

CODAP80 is an enhanced IBM version of the Comprehensive Occupational Data Analysis Programs. The software system is used to process occupational information and includes programs that range from data entry to statistical analysis. CODAP80 is based on a database management concept which allows the job analyst more versatility in analysis than its predecessor. Included with the system are four manuals: the CODAP80 User's Manual, Job Analysis Manual, Systems Manual, and Executive Summary.

kp.1

17. Document Analysis a. Descriptors

1,

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement:
RELEASE UNLIMITED

19. Security Class (This Report)
UNCLASSIFIED

20. Security Class (This Page)
UNCLASSIFIED

21. No. of Pages
192

22. Price
[redacted]

(See ANSI-Z39.18)

See Instructions on Reverse

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. DOD/DF-84/006a	2. BIN-601	3. Recipient's Accession No.

4. Title and Subtitle

COMPREHENSIVE OCCUPATIONAL DATA ANALYSIS PROGRAMS 80 (CODAP80)
User's Manual

5. Report Date
JANUARY 1984

6.

7. Author(s)
N/A

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

NAVY OCCUPATIONAL DEVELOPMENT AND ANALYSIS CENTER (NODAC)
BUILDING 150, WASHINGTON NAVY YARD (ANACOSTIA)
WASHINGTON, DC 20374

10. Project/Task/Work Unit No.

11. Contract(C) or Grant(G) No.
(C) N/A
(G) N/A

12. Sponsoring Organization Name and Address

NAVY OCCUPATIONAL DEVELOPMENT AND ANALYSIS CENTER (NODAC)
BUILDING 150, WASHINGTON NAVY YARD (ANACOSTIA)
WASHINGTON, DC 20374

13. Type of Report & Period Covered
FINAL RELEASE 83.1

14.

15. Supplementary Notes

SOURCE CODE FOR CODAP80 PROGRAMS.
for magnetic tape see

16. Abstract (Limit: 200 words)

CODAP80 is an enhanced IBM version of the Comprehensive Occupational Data Analysis Programs. The software system is used to process occupational information and includes programs that range from data entry to statistical analysis. CODAP80 is based on a database management concept which allows the job analyst more versatility in analysis than its predecessor. Included with the system are four manuals: the CODAP80 User's Manual, Job Analysis Manual, Systems Manual, and Executive Summary.

kp.1

17. Document Analysis a. Descriptors

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement: RELEASE UNLIMITED	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 192
	20. Security Class (This Page) UNCLASSIFIED	22. Price [redacted]

(See ANSI-Z39.18) See Instructions on Reverse

FOREWORD

The Comprehensive Occupational Data Analysis Programs (CODAP), a software package developed by the United States Air Force, is in use by all the United States military services and numerous other agencies throughout the world. Of the two predominant versions of CODAP, the IBM version has not kept pace with the continuing development of the UNIVAC version.

In 1978 the Navy Occupational Development and Analysis Center, a detachment of the Naval Military Personnel Command, and serving as Executive Agent for Joint Task Analysis Support for the Department of Defense, initiated a project to develop an enhanced IBM version of CODAP which would be less machine dependent than the existing IBM version, easy for non-programmers to learn and use, and which would provide the capability to implement new analysis approaches for analyzing occupational data. The funding for this project was provided by the United States Navy, Marine Corps, and Coast Guard.

As a result of this project, CODAP80, an enhanced version of IBM CODAP, was developed by Texas A&M University. This manual is one of four CODAP80 manuals which were developed to accompany the CODAP80 system. The four manuals are the CODAP80 Executive Summary, the Job Analysis Manual, the User's Manual, and the Systems Manual.

TABLE OF CONTENTS

CODAP80

GENERAL INTRODUCTION

CODAP80 is a software system for processing occupational information. The system includes programs for basic data entry and statistical analysis. CODAP80 was designed with the particular needs of the job analyst in mind. As such, much of the system's terminology is oriented toward them. Users of CODAP80 will find, though, that the general data handling and analysis features of the system will allow any database to be processed that can be conceptualized in the form of a two-dimensional matrix.

ORGANIZATION OF THE USER'S MANUAL

from i → The User's Manual consists of two major sections: a section detailing the creation of a CODAP80 database and a section illustrating the use of the CODAP80 interpreter to process and display the information residing on the database. The database creation section of the manual will focus on the routines required to generate the database (INPSTD, OGROUP and REARNG), discuss file initialization and space requirements, detail the database creation routines' control specifications and provide a sample set of data in which to illustrate the process of constructing a CODAP80 database. The interpreter section of the user's manual will explain the use of the CODAP80 language in processing an occupational database. The function and characteristics of each of the interpreter procedures will be outlined, with examples provided to facilitate understanding.

CODAP80 RELEASE

The specifications appearing in this manual apply to release 83.1 of the CODAP80 occupational analysis computer system.

EXAMPLE JCL SETUPS

The example Job Control Language setups that appear in the manual conform to those found in Brown (1977). They should be compatible with the JCL specifications of any IBM OS operating system.

CODAP80
DATABASE CREATION

INTRODUCTION

Database creation consists of three steps: two of which are mandatory and one that is optional. The three steps are (in order):

1) INPSTD (Input Standard)
2) OGROUP (Overlap Group)
3) REARNG (Rearrange)

INPSTD

The INPSTD database creation routine builds the initial incumbent database. Raw time spent ratings are relativized to a 100 point scale and history, task and secondary remarks are processed and saved. INPSTD is a mandatory step in database creation.

OGROUP

The OGROUP database creation routine performs a hierarchical clustering of incumbents measured on their time spent on tasks. It is the main clustering routine in the CODAP80 system. OGROUP is an optional step in database creation.

REARNG

The REARNG database creation routine prepares the initial database for use by the CODAP80 interpreter. REARNG is a mandatory step in database creation.

SAMPLE DATA

A sample set of data (consisting of seven incumbents measured on four history, five task and five secondary variables) is provided to illustrate the steps in database creation. The amount of information contained in the sample set of data is small enough to allow the user to trace, by hand, the computations associated with the different steps involved in the creation of a CODAP80 database.

FILE INITIALIZATION

Before the INPSTD and OGROUP database creation routines can be run, it is necessary to initialize the file space required for their execution. A simple FORTRAN program (named INITIAL1) is provided to accomplish this. After INPSTD and OGROUP have been executed, another FORTRAN program (named INITIAL2) is provided to initialize the file space required to execute the REARNG database creation routine and the CODAP80 interpreter.

INPSTD AND OGROUP
FILE INITIALIZATION

INTRODUCTION

There are six files that must be initialized before the INPSTD and OGROUP database creation routines may be run. These six files are:

1) INPFILE
2) VARCOM
3) SYMTAB1
4) GRPFILE
5) GRPHSN
6) DECODE

Initialization of these six files is accomplished by the INITIAL1 program. INITIAL1 serves to create the necessary files, and provide them with enough space to allow INPSTD and OGROUP to execute properly.

INITIAL1

Each of the above files requires a space allocation. How much space depends on the file. The amount of space required is determined by the number of records that are written to the file. The number of records that are written is a function of the amount of the various kinds of information being input. The number of records a file should have initialized is calculated using the following equations. The number of records per track quoted assumes IBM 3350 compatible disk drives. The basic reference used is Brown (1977).

INPFILE

# Records = NINC * CEIL((NHIST + NTASK + NSEC + 2)/900)
5 Records per Track

VARCOM

# Records = (NHIST + NTASK + NSEC) +
(Potential # of Created Rows or Columns)
45 Records per Track

SYMTAB1

# Records = Always set at 31

GRPFILE

$$\# \text{ Records} = 5 + \text{FLOOR}\frac{\text{NINC} - 2}{810} + \text{FLOOR}\frac{\text{NINC} - 2}{3240} + \text{FLOOR}\frac{\text{NINC} - 1}{540}$$

1 Record per Track

GRPHSN

# Records = CEIL(NINC/10)
86 Records per Track

DECODE

# Records = 1 + # Different Ranges + CEIL(# Different Ranges/100)
140 Records per Track

In the above equations the different parameters are interpreted in the following way:

FLOOR: Largest integer <= Argument.
Example: FLOOR(6.1) = 6 FLOOR(9) = 9

CEIL: Smallest integer >= Argument.
Example: CEIL(6.1) = 7 CEIL(9) = 9

NINC: Number of incumbents in the study.

NHIST: Number of history variables.

NTASK: Number of task variables.

NSEC: Number of secondary variables.

The DECODE file equation is concerned with the "# Different Ranges." For example, the following decode titles have 7 different ranges:

```
    H15    1=YES; 2=NO;
    H16    1=LO; 2=MED; 3=HI;
H10-H20    1=HOT; 2=COLD;
```

INITIAL1
EXECUTION SETUP

On the following pages is the JCL setup and FORTRAN source code for executing the INITIAL1 routine. The procedure referenced on the "// EXEC" card (FG) is the procedure library name for the FORTRAN G1 compile, load and go procedure. The setup for FG can be found in Appendix C.

INITIAL1
JCL SETUP FOR INITIAL1 FORTRAN PROGRAM

```
//*****************************************************************
//*                                                               *
//* INITIAL1 JCL SETUP.                                           *
//* SAMPLEDATA80 DATA.                                            *
//*                                                               *
//*****************************************************************
//  EXEC  FG,REGION=256K
//FT02F001 DD DSN=INPFILE,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(DSORG=DA),SPACE=(3600,(7))
//FT10F001 DD DSN=VARCOM,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(DSORG=DA),SPACE=(244,(50))
//FT12F001 DD DSN=SYMTAB1,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(DSORG=DA),SPACE=(52,(31))
//FT15F001 DD DSN=GRPFILE,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(DSORG=DA),SPACE=(12960,(5))
//FT16F001 DD DSN=GRPHSN,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(DSORG=DA),SPACE=(40,(1))
//FT17F001 DD DSN=DECODE,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(DSORG=DA),SPACE=(120,(7))
//SOURCE   DD *

       ***** INITIAL1 FORTRAN SOURCE STATEMENTS *****

//SYSIN    DD *
```

INITIAL1
PROGRAM TO INTIALIZE THE OGROUP AND INPSTD FILES

```
C-----------------------------------------------------C
C INITIAL1 FORTRAN PROGRAM.                           C
C PROGRAM TO INITIALIZE THE FILES NECESSARY TO        C
C EXECUTE THE INPSTD AND OGROUP DATABASE CREATION     C
C ROUTINES. -- SAMPLEDATA80 DATA.                     C
C-----------------------------------------------------C
      REAL INPFIL(900),  VARCOM(61), SYMTB1(13), GRPFIL(3240),
     +     GRPHSN(10),   DECODE(30)
      DEFINE FILE  2 (   7,  900,U,IREC)
      DEFINE FILE 10 (  50,   61,U,IREC)
      DEFINE FILE 12 (  31,   13,U,IREC)
      DFFINE FILE 15 (   5, 3240,U,IREC)
      DEFINE FILE 16 (   1,   10,U,IREC)
      DEFINE FILE 17 (   7,   30,U,IREC)
C-----------------------------------------------------
C  WRITE INITIALIZATION RECORDS TO INPFILE  (FT02)
C-----------------------------------------------------
      DO 20 J=1,7
      DO 10 K=1,900
   10 INPFIL(K)=J
      IREC=J
   20 WRITE ( 2'IREC) INPFIL
C-----------------------------------------------------
C  WRITE INITIALIZATION RECORDS TO VARCOM   (FT10)
C-----------------------------------------------------
      DO 40 J=1,50
      DO 30 K=1,61
   30 VARCOM(K)=J
      IREC=J
   40 WRITE (10'IREC) VARCOM
C-----------------------------------------------------
C  WRITE INITIALIZATION RECORDS TO SYMTAB1  (FT12)
C-----------------------------------------------------
      DO 60 J=1,31
      DO 50 K=1,13
   50 SYMTB1(K)=J
      IREC=J
   60 WRITE (12'IREC) SYMTB1
C-----------------------------------------------------
C  WRITE INITIALIZATION RECORDS TO GRPFILE  (FT15)
C-----------------------------------------------------
      DO 80 J=1,5
      DO 70 K=1,3240
   70 GRPFIL(K)=J
      IREC=J
   30 WRITE (15'IREC) GRPFIL
C-----------------------------------------------------
C  WRITE INITIALIZATION RECORDS TO GRPHSN   (FT16)
C-----------------------------------------------------
      DO 100 J=1,1
      DO  90 K=1,10
   90 GRPHSN(K)=J
      IREC=J
  100 WRITE (16'IREC) GRPHSN
C-----------------------------------------------------
C  WRITE INITIALIZATION RECORDS TO DECODE   (FT17)
C-----------------------------------------------------
      DO 120 J=1,7
      DO 110 K=1,30
  110 DECODE(K)=J
      IREC=J
  120 WRITE (17'IREC) DECODE
      STOP
      END
```

INPSTD

INTRODUCTION

INPSTD is the first step in the creation of a CODAP80 database. This step is characterized by the assignment of a database or study identification designation, variable (history, task and secondary) remark and decode title specifications and, in card image form, raw incumbent data. INPSTD consists of five sections:

1) Database Parameter Specification
2) Format Fields Specification
3) Variable Remark Specification
4) Decode Title Specification
5) Incumbent Data

DATABASE PARAMETER SPECIFICATION

The database parameter specification provides the INPSTD routine with information pertaining to the size of the study to be processed. The database parameter specifications are made on a single card requesting the following information:

CARD COLUMNS

1-12 Database (or Study ID). The ID must be left justified, may only begin with a letter or underscore, contain no imbedded blanks and must consist of no characters other than A-Z, 0-9 and the underscore. The ID may be from 1 to 12 characters long.

13-17 Number of incumbents to be stored on the database (maximum of 20,000).

18-20 Number of data cards for each incumbent (maximum of 455).

21-24 Number of history (H) or background variables.

25-28 Number of task (T) variables.

29-32 Number of secondary (S) variables.

33-33 Incumbent data print indicator. If blank, incumbent data will not be printed. Any character other than a blank will cause data to be printed.

35-35 Put a "1" to suppress default relativization of task data.

36-36 Put a "1" to suppress default error check of non-filled data. If the user has placed a "1" in column 36, data will be interpreted as follows:

Assume that age of incumbent is specified on the format field as a two digit response (H.). An age designation of 2b̸ (the number 2 followed by a blank) will be interpreted as 20. If column 36 is left blank, data designations that are not right justified will stop processing with an error.

37-37 Put a "1" to suppress printing of error messages caused by non-filled data (only valid if column 36 is a "1").

The maximum number of incumbents that INPSTD can process is 20,000. The maximum number of history, task and secondary variables is 5,000. These 5,000 variables may be comprised of any combination of history, task or secondary responses.

It is not required that three types of variable responses occur in the data of an occupational investigation. An investigation may entirely consist of history responses only, or for that matter, may entirely consist of task or secondary responses. There will be a difference, though, in the way INPSTD interprets the various types of responses. Task responses will be relativized to a 0-100 point scale, while history and secondary responses will be stored in exactly the form in which they were input. In addition, a blank field input as a task response will be interpreted as a zero, while blank fields input as history or secondary responses will be interpreted by the system as being missing values.

FORMAT FIELDS SPECIFICATION

Format fields consist of H, T and S designations respectively associated with the data type response made by the incumbent. An H field designation indicates that the associated data response from the incumbent represents a history variable. A T field designation represents a task response and a S field designation represents a secondary response. If a data field consists of more than one digit, the length of the field is expressed by continuing the H, T or S designation with periods (.). For example, assume an incumbent's responses consisted of the following:

	CARD COLUMNS	DATA TYPE
Incumbent ID	1-2	History
Age	3-4	History
Sex	5-5	History
Task 1	6-6	Task
Task 2	7-7	Task
Task 3	8-8	Task

The format field that would designate such data would look like this:

```
 --- Column 1
|
H.H.HTTT
```

The maximum length of a data field is seven digits. The maximum number of format field specification cards is 455 (this would allow the specification of up to 5,000 seven digit fields). Format field specifications may not be continued across card image boundaries. The format fields specification cards are followed by a card containing an '@@' delimiter in columns 1-2.

VARIABLE REMARK SPECIFICATION

A variable remark specification is a user supplied description or definition explaining the purpose or function of an associated data item. Variable remarks are stored by INPSTD for later reference by the CODAP80 interpreter.

For every H, T or S field denoted in the format fields specification, there must be an associated variable remark specification. For instance, if five task fields were indicated in the format fields specification, then five task variable remark specifications must be present.

The form of a variable remark is:

1) Variable type indication (H, T or S).
2) Digit (an integer number appended to the varible type indication).
3) Assignment operator (the symbol '=').
4) Variable remark (user supplied text describing the associated variable).
5) Variable remarks are terminated by a semicolon (the symbol ';').

Variable remark specifications are made to the INPSTD database creation routine by placing the variable type indication in column 1, the digit identifying the history, task or secondary response in columns 2-5 (left-justified), the assignment operator in column 6 and the variable remark in columns 7-66. For example, assume the eighth task in an inventory read:

```
ESTABLISH STANDARDS OF TERMINOLOGY AND DOCUMENTATION
IN WRITING FORTRAN COMPUTER PROGRAMS.
```

The user, though, desires that the eighth task be printed-out by the CODAP80 interpreter as:

```
ESTABLISH STANDARDS OF TERMINOLOGY AND DOCUMENTATION IN
  WRITING FORTRAN COMPUTER PROGRAMS.
```

To achieve this output format, the eighth task would need to be formatted at INPSTD time in the following way:

```
--  CARD COLUMNS
|
|            1         2         3         4         5         6         7         8
--> 12345678901234567890123456789012345678901234567890123456789012345678901234567890

    T8   =ESTABLISH STANDARDS OF TERMINOLOGY AND DOCUMENTATION IN
          WRITING FORTRAN COMPUTER PROGRAMS.;
```

Variable remarks may consist of up to 240 characters. Immediately following the assignment operator (in column 6) is the first of the allowed 240 characters. INPSTD scans each character between columns 7-67 in search of a semi-colon. If no semi-colon is found, INPSTD skips to the next card and continues scanning columns 7-67 until it finds one. If, after having scanned columns 7-67 for four cards, a semi-colon has still not been found, INPSTD will signal an error that the variable remark is more than 240 characters long (columns 7-66 equal 60 characters -- four of these would equal 240 characters). Blanks in columns 7-66 are considered valid characters. The following example remark would be interpreted by INPSTD as having 240 characters:

```
--  CARD COLUMNS
|
|            1         2         3         4         5         6         7         8
--> 12345678901234567890123456789012345678901234567890123456789012345678901234567890

    H1   =************************************************************
          ************       240 CHARACTER VARIABLE       ************
          ************               REMARK               ************
          ************************************************************;
```

Variable remark specifications must be input in the following order:

All history variable remarks.
All task variable remarks.
All secondary variable remarks.

The fact that variable remark specifications must occur in a specific order does not mean that the incumbent data must be in this order also. Incumbent data may be organized in any fashion the user desires. The numeric digits appended to the variable type indicator (H, T or S) must be in ascending sequence from 1 to n with no ommissions. Semicolons are used by INPSTD to delimit the end of a remark. They should not be used in the text of a remark. The variable remark specifications are followed by a card containing an '@@' delimiter in columns 1-2.

DECODE TITLE SPECIFICATION

Decode title specifications are useful for enhancing the readability of reports by decoding abstract number classifications into understandable English. Many variables are coded '0=yes' and '1=no' or are responded to with an even greater range of classifications. Decoding the value response classifications of a variable at INPSTD time will, in the case of the CODAP80 VARSUM procedure, make for a more interpretable report.

The form of a decode title specification is as follows:

```
VARIABLE ID
     OR              DECODE VALUE = DECODE TITLE;
VARIABLE RANGE
```

A variable ID is the letter H, T or S followed by 1-4 digits. A variable range is two variable IDs with a dash in between them. There may be only 0-5 blanks on each side of the dash. The two variable IDs must be the same type (have the same beginning letter) and the numeric portion of the first variable ID must be less than that of the second variable ID. For example, H10 - H8 is an invalid variable range. The decode title begins with the character immediately after the '=' and ends with the character immediately preceding the ';'. Decode titles can be from 1-32 characters. If another decode title is to be specified for the same variable ID or variable range, it may be done by following the semicolon with:

DECODE VALUE = DECODE TITLE;

As many of these as needed may be specified for a particular variable. For example, the following is valid:

```
H5-H10  1=THE TITLE FOR DECODE VALUE 1;
        2=THE TITLE FOR DECODE VALUE 2;
        7=THE TITLE FOR DECODE VALUE 7;
```

Notice that each successive decode value must be greater than the previous one for the same variable. Variable IDs must be in ascending order and decode values must be in ascending order within variable IDs or variable ranges. The above example indicates that whenever one of the variables H5, H6, H7, H8, H9, or H10 has a value of 1, the associated meaning of that value is the decode title (in this case, it is 'THE TITLE FOR DECODE VALUE 1'). The same holds true for decode values 2 and 7.

The variable decode specifications are followed by a card containing an '@@' delimiter in columns 1-2.

CONTROL

The four INPSTD sections discussed above make-up the control portion of the routine. A sample set of incumbent information has been prepared to guide the user through this manual. The information consists of seven incumbents, each measured on four history, five task and five secondary indicies. Using this sample information, the control setup of the INPSTD routine would be as shown on the following page.

CONTROL SETUP

```
 -- CARD COLUMNS
|
|            1         2         3         4         5         6         7         8
 --> 12345678901234567890123456789012345678901234567890123456789012345678901234567890

    SAMPLEDATA8000007001000400050005Y
    HH.H.HTSTSTSTSTS
    @@
    H1   =SEX;
    H2   =AGE;
    H3   =YEARS ON JOB;
    H4   =INCUMBENT ID;
    T1   =SUBDUE VIOLENT INMATES;
    T2   =SHAKE DOWN INMATES;
    T3   =SHAKE DOWN VISITORS;
    T4   =ESCORT INMATES;
    T5   =TESTIFY IN COURT;
    S1   =SECONDARY - SUBDUE VIOLENT INMATES;
    S2   =SECONDARY - SHAKE DOWN INMATES;
    S3   =SECONDARY - SHAKE DOWN VISITORS;
    S4   =SECONDARY - ESCORT INMATES;
    S5   =SECONDARY - TESTIFY IN COURT;
    @@
    H1  1=MALE; 2=FEMALE;
    S1-S5 1=DO; 2=ASSIST; 3=SUPERVISE;
    @@
```

INCUMBENT DATA

Each incumbent's data may consist of up to 455 cards. All 80 characters of a card may contain data. The maximum length of a data field is seven characters. The maximum number of variable responses from an incumbent is 5,000. The maximum number of incumbents is 20,000. Data fields may not span across cards. All data from an incumbent must be numeric. If potential task information was indicated on the database parameter specification card and an incumbent has no nonzero task information, then that incumbent is not included in the database.

DATA

The fifth INPSTD section discussed above constitutes the data portion of the routine. Using the sample information, the data portion of INPSTD would be as follows:

```
 -- CARD COLUMNS
|
|            1         2         3         4         5         6         7         8
 --> 12345678901234567890123456789012345678901234567890123456789012345678901234567890

    219 117 1111220
    14119212420 2221
    1  1630 33430 0
    127 344 41310 63
    123 251 1122510
    15330642710 0 0
    2  1170 0 225231
```

INPSTD
EXECUTION SETUP

The JCL setup necessary to execute the INPSTD database creation routine may be found on the following page. Printed output generated from INPSTD is displayed also.

INPSTD
EXECUTION JCL FOR INPSTD DATABASE CREATION ROUTINE

```
//*******************************************************
//*  JCL SETUP TO EXECUTE THE INPSTD DATABASE CREATION   *
//*  ROUTINE.  INPSTD IS STORED AS A MEMBER IN PDS LOAD *
//*  MODULE CODAP80.                                     *
//*******************************************************
//  EXEC  PGM=INPSTD,REGION=512K
//STEPLIB  DD DSN=CODAP80,DISP=SHR
//FT02F001 DD DSN=INPFILE,DISP=OLD
//FT03F001 DD DSN=CONTROL,DISP=OLD
//FT04F001 DD DSN=DATA,DISP=OLD
//FT06F001 DD SYSOUT=A
//FT10F001 DD DSN=VARCOM,DISP=OLD
//FT12F001 DD DSN=SYMTAB1,DISP=OLD
//FT17F001 DD DSN=DECODE,DISP=OLD
//FT29F001 DD DSN=&&TEMP13,UNIT=SYSDA,DISP=(NEW,DELETE),
//            DCB=(RECFM=F,LRECL=48,BLKSIZE=48),
//            SPACE=(48,(5000,1))
```

CONTENTS OF DSN=CONTROL

```
SAMPLEDATA80000070010004000050005Y
HH.H.HTSTSTSTSTS
@@
H1   =SEX;
H2   =AGE;
H3   =YEARS ON JOB;
H4   =INCUMBENT ID;
T1   =SUBDUE VIOLENT INMATES;
T2   =SHAKE DOWN INMATES;
T3   =SHAKE DOWN VISITORS;
T4   =ESCORT INMATES;
T5   =TESTIFY IN COURT;
S1   =SECONDARY - SUBDUE VIOLENT INMATES;
S2   =SECONDARY - SHAKE DOWN INMATES;
S3   =SECONDARY - SHAKE DOWN VISITORS;
S4   =SECONDARY - ESCORT INMATES;
S5   =SECONDARY - TESTIFY IN COURT;
@@
H1  1=MALE; 2=FEMALE;
S1-S5 1=DO; 2=ASSIST; 3=SUPERVISE;
@@
```

CONTENTS OF DSN=DATA

```
219 117 1111220
14119212420 2221
1  1630 33430 0
127 344 41310 63
123 251 1122510
15330642710 0 0
2  1170 0 225231
```

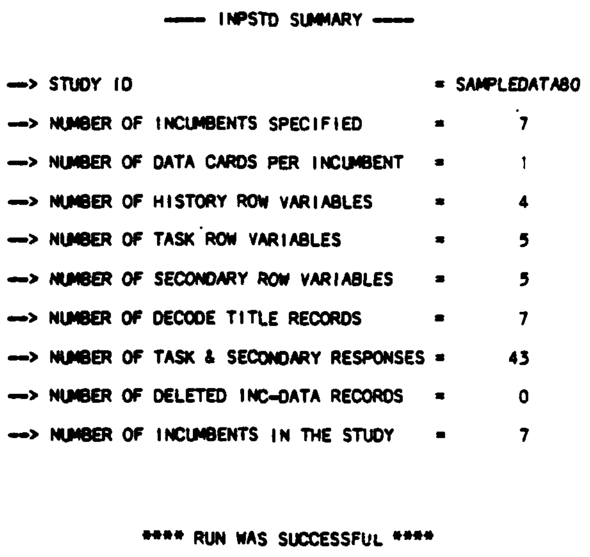
INPSTD
PRINTED OUTPUT

```
                              VARIABLE FORMAT SPECIFICATION

                   1         2         3         4         5         6         7         8
          12345678901234567890123456789012345678901234567890123456789012345678901234567890

          SAMPLEDATA80    7  1   4   5   5Y

CARD
NUMBER

     1  HH,H,HTSTSTSTSTS


                                   H-TYPE   VARIABLE REMARKS

VARIABLE          VARIABLE             VARIABLE
  TYPE             NUMBER              COMMENT

   H                  1       =       SEX
   H                  2       =       AGE
   H                  3       =       YEARS ON JOB
   H                  4       =       INCUMBENT ID


                                   T-TYPE   VARIABLE REMARKS

VARIABLE          VARIABLE             VARIABLE
  TYPE             NUMBER              COMMENT

   T                  1       =       SUBDUE VIOLENT INMATES
   T                  2       =       SHAKE DOWN INMATES
   T                  3       =       SHAKE DOWN VISITORS
   T                  4       =       ESCORT INMATES
   T                  5       =       TESTIFY IN COURT

                                   S-TYPE   VARIABLE REMARKS

VARIABLE          VARIABLE             VARIABLE
  TYPE             NUMBER              COMMENT

   S                  1       =       SECONDARY - SUBDUE VIOLENT INMATES
   S                  2       =       SECONDARY - SHAKE DOWN INMATES
   S                  3       =       SECONDARY - SHAKE DOWN VISITORS
   S                  4       =       SECONDARY - ESCORT INMATES
   S                  5       =       SECONDARY - TESTIFY IN COURT
```

INPSTD
PRINTED OUTPUT
(continued)

```
                                                         H-TYPE  DECODE TITLES

                               BEGINNING         ENDING
               VARIABLE         VARIABLE        VARIABLE        DECODE        DECODE
                 TYPE            NUMBER          NUMBER         VALUE         TITLE

                  H                  1.              1.             1.        MALE
                  H                  1.              1.             2.        FEMALE

                                                         S-TYPE  DECODE TITLES

                               BEGINNING         ENDING
               VARIABLE         VARIABLE        VARIABLE        DECODE        DECODE
                 TYPE            NUMBER          NUMBER         VALUE         TITLE

                  S                  1.              5.             1.        DO
                  S                  1.              5.             2.        ASSIST
                  S                  1.              5.             3.        SUPERVISE

                                                            INCUMBENT DATA

                                        1         2         3         4         5         6         7         8
                               12345678901234567890123456789012345678901234567890123456789012345678901234567890
INCUMBENT      1 :       1 --->219 117 1111220
                  NUMBER OF NON-ZERO TASKS =    4                                                                   <---

INCUMBENT      2 :       1 --->14119212420 2221
                  NUMBER OF NON-ZERO TASKS =    4                                                                   <---

INCUMBENT      3 :       1 --->1  1630 33430 0
                  NUMBER OF NON-ZERO TASKS =    2                                                                   <---

INCUMBENT      4 :       1 --->127 344 41310 63
                  NUMBER OF NON-ZERO TASKS =    4                                                                   <---

INCUMBENT      5 :       1 --->123 251 1122510
                  NUMBER OF NON-ZERO TASKS =    4                                                                   <---

INCUMBENT      6 :       1 --->15330642710 0 0
                  NUMBER OF NON-ZERO TASKS =    2                                                                   <---

INCUMBENT      7 :       1 --->2  1170 0 225231
                 .NUMBER OF NON-ZERO TASKS =    3                                                                   <---
```

INPSTD
PRINTED OUTPUT
(continued)



```
                  ----- INPSTD SUMMARY -----

 ---> STUDY ID                              = SAMPLEDATA80
 ---> NUMBER OF INCUMBENTS SPECIFIED        =        7
 ---> NUMBER OF DATA CARDS PER INCUMBENT    =        1
 ---> NUMBER OF HISTORY ROW VARIABLES       =        4
 ---> NUMBER OF TASK ROW VARIABLES          =        5
 ---> NUMBER OF SECONDARY ROW VARIABLES     =        5
 ---> NUMBER OF DECODE TITLE RECORDS        =        7
 ---> NUMBER OF TASK & SECONDARY RESPONSES =       43
 ---> NUMBER OF DELETED INC-DATA RECORDS    =        0
 ---> NUMBER OF INCUMBENTS IN THE STUDY     =        7


               **** RUN WAS SUCCESSFUL ****
```

OGROUP

INTRODUCTION

OGROUP is the main clustering routine in the CODAP80 computer system. The routine performs a hierarchical clustering (based on Ward, 1963) of incumbents measured on tasks. OGROUP is an optional step in database creation. If clustering is desired, then OGROUP should be run immediately after INPSTD and before REARNG. OGROUP consists of two sections:

1) Parameter Specification
2) Title Specification

PARAMETER SPECIFICATION

The parameter specifications for the OGROUP routine are made on a single card requesting the following information:

CARD COLUMNS	
1-12	Study ID.
14-14	Overlap equation number (see Appendix B for formulae). 1=Absolute overlap 2=Distance 3=Distance squared 4=Binary
22-22	Overlap matrix print indication. Y=Print overlap matrix.
24-24	Cluster indication. Y=Do clustering N=Clustering has been performed
26-26	Membership report indication. Y=Print a group membership report
27-27	Diagram report indication. Y=Print a diagram report
32-36	Minimum group membership for diagram.

TITLE SPECIFICATION

The title specification is made on the card immediately following the parameter specifications.

CARD COLUMNS	
1-72	Report title.

OGROUP INPUT SETUP

The parameter and title spécifications for clustering the incumbents associated with the sample data are displayed below. The user has opted to use absolute overlap as the similarity formula, has indicated that clustering is to be performed, a group membership report is to be made and a diagram report is to be generated with a minimum starter group membership of 2.

```
— CARD COLUMNS
|
|            1         2         3         4         5         6         7         8
—> 12345678901234567890123456789012345678901234567890123456789012345678901234567890

   SAMPLEDATA80 1         Y YY        2
   CLUSTERING INCUMBENTS — SAMPLE DATABASE — N=7 — TASKS=5
```

OGROUP RESOURCE CONSIDERATIONS

There is, theoretically, no limit to the number of incumbents that may be clustered with OGROUP. The user should keep in mind, though, that the time it takes to run OGROUP is a function of the number of tasks multiplied by the square of the number of incumbents.

OGROUP EXECUTION SETUP

The setup necessary to execute the OGROUP database creation routine may be found on the following page. Output generated from OGROUPS's execution is displayed also.

OGROUP
EXECUTION JCL FOR OGROUP DATABASE CREATION ROUTINE

```
//**********************************************************
//*  JCL SETUP TO EXECUTE THE OGROUP DATABASE CREATION   *
//*  ROUTINE.  OGROUP IS STORED AS A MEMBER IN PDS LOAD  *
//*  MODULE CODAP80.                                      *
//**********************************************************
//  EXEC PGM=OGROUP,REGION=512K
//STEPLIB  DD DSN=CODAP80,DISP=SHR
//FT02F001 DD DSN=INPFILE,DISP=OLD
//FT05F001 DD DDNAME=SYSIN
//FT06F001 DD SYSOUT=A
//FT12F001 DD DSN=SYMTAB1,DISP=OLD
//FT15F001 DD DSN=GRPFILE,DISP=OLD
//FT16F001 DD DSN=GRPHSN,DISP=OLD
//FT21F001 DD DSN=&&TEMP21,UNIT=SYSDA,DISP=(NEW,DELETE),
//            DCB=(RECFM=F,LRECL=12960,BLKSIZE=12960),
//            SPACE=(12960,(4000,1))
//SYSIN    DD *
SAMPLEDATA80 1         Y YY       2
CLUSTERING INCUMBENTS -- SAMPLE DATABASE -- N=7 -- TASKS=5
```

SCRATCH FILE CALCULATION

Temporary scratch file FT21F001 in the above JCL setup will request 4000 records, each 12960 bytes long. This would allow up to 3350 incumbents to be clustered, each measured on up to 1000 tasks (the file would need 3926 records). This amount of scratch space is not always necessary. To calculate the amount of space needed, use the following equation:

```
# Records = 12 + FLOOR(NINC/1620) + FLOOR((NINC-1)/1568)
            + FLOOR((NINC-2)/1080) + FLOOR((NINC-2)/810)
            + FLOOR((NINC*NTASK)/1620) + FLOOR((NINC-1)/540)
            + (1 + FLOOR((NINC-1)/56))**2
            - (FLOOR((NINC-1)/56) * (1 + FLOOR((NINC-1)/56)))/2
```

In the above equation the different parameters are interpreted in the following way:

FLOOR: Largest integer <= Argument.
Example: FLOOR(2.1) = 2 FLOOR(3) = 3

NINC: Number of incumbents in the study.

NTASK: Number of task variables.

If the number of records in FT21F001 needs to be increased or decreased, check with your CODAP80 installation representative.

OGROUP
PRINTED OUTPUT

```
STUDY ID                           = SAMPLEDATA80

OVERLAP EQUATION NUMBER            =       1

YES/NO OPTIONS     2         3
               01234567890
                   Y YY

MINIMUM GROUP MEMBERSHIP FOR DIAGRAM =     2

REPORT TITLE                       = CLUSTERING INCUMBENTS -- SAMPLE DATABASE -- N=7 -- TASKS=5


                     RESULTANT GROUP           FORMED BY                       COMBINING GROUPS
             GROUP   NO.    SEQUENCE      AVERAGE                    NUM   AT     SEQUENCE              NUM   AT     SEQUENCE
     STAGE   IDENT  MBERS  FROM   TO    BETWEEN   WITHIN     IDENT  MBRS  STAGE  FROM   TO      IDENT  MBRS  STAGE  FROM   TO
       1       5      2      2     3    70.0000   70.0000      5     1      0     0     0        7     1      0     0     0
       2       2      2      4     5    56.8627   56.8627      2     1      0     0     0        4     1      0     0     0
       3       2      3      4     6    51.3072   53.1590      2     2      2     4     5        6     1      0     0     0
       4       1      3      1     3    47.4747   54.9831      1     1      0     0     0        5     2      1     2     3
       5       2      4      4     7    42.2969   47.7279      2     3      3     4     6        3     1      0     0     0
       6       1      7      1     7    38.3838   43.4249      1     3      4     1     3        2     4      5     4     7


CLUSTERING INCUMBENTS -- SAMPLE DATABASE -- N=7 -- TASKS=5                  MINIMUM GROUP MEMBERSHIP =     2

  1 2       2 2
  2-3       4-5
70.  70.  56.  56.
    |         |
    |         |
  4 3       3 3
  1-3       4-6
47.  55.  51.  53.
    |         |
    |         |
    |       5 4
    |       4-7
    |     42.  47.
    |         |
  6 7         |
  1-7  ---------
38.  43.

**** END OF RUN ****
```

CARDFILE

The CARDFILE is a card image sequential file. INITIAL2 will initialize it with one record. It is used by the CODAP80 interpreter procedure COPY. To determine the number of records that will be written to CARDFILE the user is referred to the discussion of the COPY procedure.

In the equations on the previous page the different parameters are interpreted in the following way:

CEIL: Smallest integer >= Argument.
Example: CEIL(6.1) = 7 CEIL(9) = 9

NINC: Number of incumbents in the study.

NHIST: Number of history variables.

NTASK: Number of task variables.

NSEC: Number of secondary variables.

INITIAL2
EXECUTION SETUP

On the following page is the JCL setup and FORTRAN source code for executing the INITIAL2 routine. The procedure referenced on the "// EXEC" card (FG) is the procedure library name for the FORTRAN G1 compile, load and go procedure. The setup for FG can be found in Appendix C.

REARNG AND INTERPRETER FILE INITIALIZATION

INTRODUCTION

Following the execution of INPSTD and OGROUP, four files must be initialized before the REARNG database creation routine and the CODAP80 interpreter may be run. These four files are:

1) DATABASE
2) CREATED
3) SYMTAB2
4) CARDFILE

Initialization of these four files is accomplished by the INITIAL2 program.

INITIAL2

Each of the above files requires a space allocation. The amount of space to be allocated is a function of the number of records that need to be written to the file. The number of records needed is calculated from the following equations. The number of records per track quoted assumes IBM 3350 compatible disk drives. The basic reference is Brown (1977).

DATABASE

```
# Records = CEIL((NHIST * NINC)/1000)
          + 2 * CEIL((# of Task & Secondary Responses)/1000)
          + CEIL((NTASK * 2)/1000)
          + CEIL((NSEC * 2)/1000)
          + 2 * CEIL((NINC - 1)/1000)
          + 2 * CEIL(NINC/1000)
4 Records per Track
```

The # of Task and Secondary Responses is found on the INPSTD summary page printed at the end of the INPSTD output.

CREATED

```
# Records = 4 + (# Potential 1000 Element Created Rows/Columns)
4 Records per Track
```

SYMTAB2

```
# Records = 120 @ 960 Bytes per
16 Records per Track = 8 Tracks
```

INITIAL2
PROGRAM TO INITIALIZE THE REARNG AND INTERPRETER FILES

```
//**************************************************
//*                                                *
//* JCL SETUP.  INITIAL2 PROGRAM.                  *
//* SAMPLEDATA80 DATA.                             *
//*                                                *
//**************************************************
//  EXEC  FG,REGION=256K
//FT01F001 DD DSN=DATABASE,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(DSORG=DA),SPACE=(4000,(9))
//FT08F001 DC DSN=CREATED,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(DSORG=DA),SPACE=(4000,(10))
//FT13F001 DD DSN=SYMTAB2,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(DSORG=DA),SPACE=(960,(120))
//FT14F001 DD DSN=CARDFILE,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),
//             SPACE=(TRK,(1))
//SOURCE   DC *
C-------------------------------------------------C
C INITIAL2 PROGRAM.                               C
C PROGRAM TO INITIALIZE THE FILES NECESSARY TO    C
C EXECUTE THE REARNG DATABASE CREATION ROUTINE AND C
C THE CODAP80 INTERPRETER.  SAMPLEDATA80 DATA.    C
C-------------------------------------------------C
      REAL DATABS(1000), CREATE(1000), SYMTB2(240)
      DEFINE FILE  1 (  9, 1000,U,IREC)
      DEFINE FILE  8 ( 10, 1000,U,IREC)
      DEFINE FILE 13 (120,  240,U,IREC)
C---------------------------------------
C  WRITE INITIALIZATION RECORDS TO DATABASE
C---------------------------------------
      DO 140 J=1,9
      DO 130 K=1,1000
  130 DATABS(K) =J
      IREC=J
  140 WRITE ( 1'IREC) DATABS
C---------------------------------------
C  WRITE INITIALIZATION RECORDS TO CREATED
C---------------------------------------
      DO 160 J=1,10
      DO 150 K=1,1000
  150 CREATE(K)=J
      IREC=J
  160 WRITE ( 8'IREC) CREATE
C---------------------------------------
C  WRITE INITIALIZATION RECORDS TO SYMTAB2
C---------------------------------------
      DO 180 J=1,120
      DO 170 K=1,240
  170 SYMTB2(K)=J
      IREC=J
  180 WRITE (13'IREC) SYMTB2
C---------------------------------------
C  WRITE INITIALIZATION RECORD TO CARDFILE
C---------------------------------------
      REWIND14
      WRITE (14,9999)
 9999 FORMAT(9HCARD FILE)
      STOP
      END
//SYSIN   DD *
```

REARNG

INTRODUCTION

REARNG (Data Rearrangement) alters the physical structure of the database to allow more efficient access to and retrieval of data items. Before data rearrangement occurs the physical records of the files making up the database hold information about many variables for one incumbent. After data rearrangement they hold information about many incumbents for one variable.

REARNG is the last step in preparing the database for use by the CODAP80 interpreter. In most studies, REARNG will immediately follow OGROUP. If OGROUP is omitted, then REARNG will follow INPSTD.

NOTE

The data access routines in the CODAP80 interpreter assume that the data are physically laid out as they will be after successful execution of REARNG. The interpreter will not function if the database has not been rearranged.

If the OGROUP routine has been executed, the REARNG routine will store the incumbents in hierarchical sequence order. If the OGROUP routine has not been run, the incumbents will be stored in the order they were originally received by INPSTD.

NOTE

There are no control cards for the REARNG routine.

REARNG EXECUTION SETUP

The setup necessary to execute the REARNG database creation routine may be found on the following page. The only output generated by REARNG is whether or not the run was successful.

REARNG
EXECUTION JCL FOR REARNG DATABASE CREATION ROUTINE

```
//********************************************************
//*                                                      *
//*  JCL SETUP TO EXECUTE THE REARNG DATABASE CREATION  *
//*  ROUTINE.  REARNG IS STORED AS A MEMBER IN PDS LOAD  *
//*  MODULE CODAP80.                                     *
//*                                                      *
//********************************************************
//  EXEC  PGM=REARNG,REGION=512K
//STEPLIB  DD DSN=CODAP80,DISP=SHR
//FT01F001 DD DSN=DATABASE,DISP=OLD
//FT02F001 DD DSN=INPFILE,DISP=OLD
//FT06F001 DD SYSOUT=A
//FT12F001 DD DSN=SYMTAB1,DISP=OLD
//FT13F001 DD DSN=SYMTAB2,DISP=OLD
//FT15F001 DD DSN=GRPFILE,DISP=OLD
//FT16F001 DD DSN=GRPHSN,DISP=OLD
//FT24F001 DD DSN=&&TEMP24,UNIT=SYSDA,DISP=(NEW,DELETE),
//            DCB=(RECFM=F,LRECL=3600,BLKSIZE=3600),
//            SPACE=(3600,(5000,1))
```

THE CODAP80 INTERPRETER

INTRODUCTION

Up to this point, the reader has been made .familiar with the database creation phase of CODAP80. The rest of this manual is to familiarize the reader with the phase of CODAP80 that will actually generate the summary statistics and reports necessary in the analysis of occupational information.

The following portion of this manual will cover the principles of the CODAP80 language, the building blocks by which CODAP80 language statements are written, the individual procedures existing in the CODAP80 system for manipulating the database and producing reports and will illustrate, through the use of syntax graphs (a kind of map detailing the syntax keywords allowed in a procedure and the organizational path of proper procedure statement construction) and examples, the method by which the system is used as a tool by the job analyst.

At the present stage of development, there are 17 procedures for data base manipulation and reporting residing in the CODAP80 system.

These 17 procedures are:

ADDATA
AVALUE
BEGIN
CLUSTER
COPY
CORR
CREATE
DESCRIBE
END
INPUT
PRINT
RANDOM
RELY
REPORT
SELECT
STANDARD
VARSUM

The function, syntax and options of each of these procedures are detailed. Each of the CODAP80 interpreter procedures are presented in alphabetical order. Appendix A contains a complete CODAP80 interpreter run sequence as would be submitted to the computer by a user. The presentation order of the procedures appearing in this run sequence are loosely organized in a manner consistent with the steps traditionally taken in job analytic studies. It should be recognized though that CODAP80 represents a significant departure from previous occupational data analysis computer systems, both in the power that can be brought to bear on the occupational database and in the conceptualization of how the job analyst goes about answering questions of occupational information.

PRINCIPLES OF THE CODAP80 LANGUAGE

WHY A LANGUAGE?

CODAP80 is a specialized and extended database management system. It stores, retrieves, and processes data in an organized and systematic manner without redundancies. As such, some facility is needed to instruct the system exactly which of its many functions should be performed in a given situation.

Many database management systems use a data manipulation language as the vehicle for communication between the user and the system. This idea was borrowed for CODAP80, and then extended. The CODAP80 language does indeed give the job analyst most of the functions that a data manipulation language would, but it goes farther by also providing statistical and analytical procedures unique to job analysis in their current application.

DESIGN GOALS

The CODAP80 language was designed with several purposes in mind, including:

- providing the job analyst with a powerful tool for accomplishing general data management
- providing the job analyst with a computerized capability to carry out any known or foreseeable type of job-analysis related data processing
- providing the job analyst with a simple, easy to learn, and easy to use method for performing basic job analysis.

ENGLISH-LIKE SYNTAX & SEMANTICS

The CODAP80 language is English-like. It can be used to write programs of statements that resemble sentences. A reader who understands the basic function of the action invoked by each CODAP80 verb can read through a program written by someone else and still gain a good understanding of what the program did.

CODAP80 statements begin with a verb or other major key word describing the action to be performed.

Variable names and options appear as qualifiers in the CODAP80 statement.

Each statement ends with a period.

Commas and blanks are used as delimiters between keywords and other segments of the statements.

FREE FORMAT

CODAP80 statements in general may appear anywhere on the card image. Card column dependencies have been avoided in the language. Statements may even cross card image boundaries.

HOW USED

To cause the CODAP80 system to perform job analysis functions, the job analyst prepares a CODAP80 source program with statements in the proper order to specify the desired actions, then submits the CODAP80 source program to the CODAP80 interpreter. The interpreter will validate the program, translate it into appropriate internal representation, and execute it, generating reports and manipulating data as specified by the program.

CODAP80 IDs

Created IDs, that is, IDs that are added to the database after database creation, may be up to 12 characters long. The first character must be a letter or underscore and contain no special characters ($, &, @, #, -, etc.) other than the underscore.

RESERVED WORDS

Certain words or sets of characters may not be used as created IDs. The words in question are already being used by CODAP80 as part of its working vocabulary. See the section on CODAP80 reserved words to identify which ones they are.

CODAP80 LIMITS

No more than 300 created IDs may occur in a single CODAP80 run stream. No more than 2000 unique words may appear in a CODAP80 source program.

SYNTAX ERROR HANDLING

CODAP80's interpreter scans the language source input by the user and, if a procedure statement is used improperly, or a keyword is misspelled, flags the error with a dollar sign ('$'), changes the status of the run to one of syntax checking only, and prints out an appropriate error message (if possible). If even one error is detected, the CODAP80 source language statements will not be executed.

As an example, note the following source language statements:

```
BEGIN SAMPLEDATA80 EXECUTE.
PRINT COLUMNS (INCMBENTS) NOREMARKS / ROWS (H1-H5)
   HEADING='PRINT COMMAND CONTAINING SYNTAX ERRORS'.
END.
```

CODAP80 will respond to the errors (INCUMBENTS spelled wrong, and there is no H5 on the database) in the above statements in the following way.

```
BEGIN SAMPLEDATA80 EXECUTE.
PRINT COLUMNS (INCMBENTS) NOREMARKS / ROWS (H1-H5)
                        $
ERROR MESSAGE  25
                                                $
ERROR MESSAGE   9
                                                 $
ERROR MESSAGE  40
   HEADING='PRINT COMMAND CONTAINING SYNTAX ERRORS'.
END.

****************************************************
ERROR FOUND IN SOURCE CODE-EXECUTION PHASE CANCELLED
****************************************************

 ERROR MESSAGES

    9  INTEGER PORTION OF SYSTEM VARIABLE TOO LARGE
   25  A GROUP NAME HAS NOT BEEN SPECIFIED
   40  EXPECTING HISTORY VARIABLE IN SEQUENCE
```

CODAP80 INTERPRETER EXECUTION SETUP

A complete CODAP80 interpreter run stream, with JCL setup, may be found on the following page. Note that some of the files that were necessary during database creation no longer appear.

CODAP80 INTERPRETER
EXECUTION JCL FOR THE CODAP80 INTERPRETER

```
//**********************************************************
//*  JCL SETUP TO EXECUTE THE CODAP80 INTERPRETER.  THE    *
//*  INTERPRETER (INTERP) IS STORED AS A MEMBER IN PDS     *
//*  LOAD MODULE CODAP80.                                  *
//**********************************************************
//  EXEC  PGM=INTERP,REGION=820K
//STEPLIB  DD DSN=CODAP80,DISP=SHR
//FT01F001 DD DSN=DATABASE,DISP=OLD
//FT05F001 DD DDNAME=SYSIN
//FT06F001 DD SYSOUT=A
//FT07F001 DD SYSOUT=B
//FT08F001 DD DSN=CREATED,DISP=OLD
//FT10F001 DD DSN=VARCOM,DISP=OLD
//FT13F001 DD DSN=SYMTAB2,DISP=OLD
//FT14F001 DD DSN=CARDFILE,DISP=OLD
//FT17F001 DD DSN=DECODE,DISP=OLD
//FT18F001 DD DSN=ERRORFIL,DISP=SHR
//FT20F001 DD DSN=&&TEMP20,UNIT=SYSDA,DISP=(NEW,DELETE),
//            DCB=(RECFM=F,LRECL=3600,BLKSIZE=3600),
//            SPACE=(3600,(6000,1))
//FT21F001 DD DSN=&&TEMP21,UNIT=SYSDA,DISP=(NEW,DELETE),
//            DCB=(RECFM=F,LRECL=12960,BLKSIZE=12960),
//            SPACE=(12960,(1550,1))
//FT22F001 DD DSN=&&TEMP22,UNIT=SYSDA,DISP=(NEW,DELETE),
//            DCB=(RECFM=F,LRECL=12960,BLKSIZE=12960),
//            SPACE=(12960,(1550,1))
//FT23F001 DD DSN=&&TEMP23,UNIT=SYSDA,DISP=(NEW,DELETE),
//            DCB=(RECFM=F,LRECL=244,BLKSIZE=244),
//            SPACE=(244,(1000,1))
//FT24F001 DD DSN=&&TEMP24,UNIT=SYSDA,DISP=(NEW,DELETE),
//            DCB=(RECFM=F,LRECL=3600,BLKSIZE=3600),
//            SPACE=(3600,(6000,1))
//FT25F001 DD DSN=&&TEMP25,UNIT=SYSDA,DISP=(NEW,DELETE),
//            DCB=(RECFM=F,LRECL=3600,BLKSIZE=3600),
//            SPACE=(3600,(6000,1))
//FT26F001 DD DSN=&&TEMP26,UNIT=SYSDA,DISP=(NEW,DELETE),
//            DCB=(RECFM=F,LRECL=4000,BLKSIZE=4000),
//            SPACE=(4000,(200,1))
//SYSIN    DD *
BEGIN SAMPLEDATA80 EXECUTE.
CLUSTER COLUMNS INCUMBENTS FOR TASKS OVL MAXIMIZE
   INCHSN NOSAVE 'INCUMBENT HSN'
   MINMEM=2
   HEADING='INCUMBENT CLUSTERING'.
ADDATA ROWS FOR INCUMBENTS N=5
   TRACTOR       NOSAVE 'OPERATE TRACTOR'
   JACKHAMMER    NOSAVE 'OPERATE JACKHAMMER'
   BULLDOZER     NOSAVE 'OPERATE BULLDOZER'
   POWERWRENCH   NOSAVE 'OPERATE POWERWENCH'
   FLAMETHROWER  NOSAVE 'OPERATE FLAMETHROWER' FORMAT '(7F1.0)'.
SELECT ROWS NEWDUTY (H4 TRACTOR JACKHAMMER BULLDOZER POWERWENCH FLAMETHROWER
   INCHSN) NOSAVE 'NEWDUTY'.
PRINT ROWS (NEWDUTY) / COLUMNS (INCUMBENTS) MISSING
   HEADING='THE CODAP80 INTERPRETER'.
END.
1100011
0010100
1100000
1001000
0000001
```

CODAP80 RESERVED WORDS
THE FOLLOWING WORDS MAY NOT BE USED AS CREATED IDS

ADDATA
ADJUST
ALL
AVALUE
AVE
AVGA
AVGP
B
BEGIN
BINARY
BY
CARD
CCNST
CCOLS
CGRPS
CLUSTER
CMODS
COL
COLS
COLUMN
COLUMNS
CONSTANTS
COPY
CORR
COUNT
CREATE
CROWS
CUM
D
DECODE
DES
DESCEND
DESCENDING
DESCRIBE
DIAGRAM
DISTANCE
DSQUARE
D2
ELSE
END
EXECUTE
FOR
FORMAT
FROM
GROUPS
HEADING
HROWS
HSN
HVARS
IF
IN
INCS
INCUMBENTS
INPUT
L
LIST
MAX
MAXIMIZE
MEAN
MIN
MINMEM
MISSING
MODS
MODULES
N
NHIST
NINCS
NONZERO
NOPAGE
NOREM
NOREMARKS
NORESET
NOSAVE
NOSKIP
NOSTID
NOSUMMARY
NOT
NSEC
NTASK
ON
OVERLAP
OVL
OVLGRP
PAGE
PCNT
PERCENT
PRINT
RANDOM
RAWSUM
REL
RELY
REPORT
RESET
ROW
ROWS
SAVE
SELECT
SORT
SQRT
SROWS
STANDARD
STAT
STD
STD
STDA
STDP
SUM
SUMONLY
SVARS
SYSCNST
SYSCOLS
SYSGROUPS
SYSMODS
SYSROWS
TAPE
TASKS
THEN
TROWS
TVARS
USING
VARSUM
WITHIN

THE SAMPLE DATABASE

INTRODUCTION

The sample database was generated through the execution of the database creation routines (INPSTD, OGROUP and REARNG). The value found in the sample database will be used in the examples given of the CODAP80 procedure language statements which appear in the rest of this manual. The sample database provides a consistent point of reference for creating meaningful examples. By examining the database, the reader should be able to determine just exactly where the numbers generated by the example CODAP80 procedure language statements came from and thereby gain a better understanding of the function of each language statement.

THE SAMPLE DATABASE

In the sample database there are seven incumbent workers; each having been asked to respond to four history variables (H1-H4), five task variables (T1-T5) and five secondary variables (S1-S5). The incumbent worker designations represent the columns of the database and the variables the incumbents are measured on represent the rows of the database. Referring to the row variable H4 (Incumbent ID), the reader will notice that the incumbents are not in the order in which they were originally input (see INPSTD). Following INPSTD, OGROUP was run on the database. When the main OGROUP clustering routine is ever executed on the database, the REARNG routine will reorganize the database by sorting the incumbents in ascending hierarchical sequence number (HSN) order as defined by the OGROUP routine (this reorganization as a function of HSN is for purely internal systems level processing - the user only needs to be aware that a reorganization has occurred). Had the main clustering routine not been run on the database, REARNG would have left the database in the order in which it was originally input.

SYSTEM CLUSTER GROUPS

The ID's G1-G6 are system cluster groups generated from the execution of the OGROUP routine. They represent the incumbent aggregates that were formed during the cluster operation. Any future reference to any of these system group ID's in CODAP80 language statements will serve to identify to the system which incumbents (columns) are to be addressed for processing. For example, were the system cluster group ID 'G4' to be referenced, the CODAP80 system would direct processing to columns 1-3 of the database.

DATABASE VALUES

All asterisks (*) occurring in the sample database indicate a missing value. Some of the values in the database have been rounded.

SAMPLE DATABASE

	I1	I2	I3	I4	I5	I6	I7
H1	2	1	2	1	1	1	1
H2	19	23	*	41	27	53	*
H3	1	2	11	19	3	30	16
H4	1	5	7	2	4	6	3
T1	64	11	0	11	24	36	0
T2	9	11	0	44	24	64	43
T3	9	22	20	0	18	0	57
T4	18	56	50	22	0	0	0
T5	0	0	30	22	35	0	0
S1	*	*	*	2	*	2	*
S2	1	1	*	2	1	1	3
S3	1	2	2	*	1	*	3
S4	2	1	2	2	*	*	*
S5	*	*	1	1	3	*	*

STUDY ID
SAMPLEDATA80

DATABASE REMARKS

H1	SEX
H2	AGE
H3	YEARS ON JOB
H4	INCUMBENT ID
T1	SUBDUE VIOLENT INMATES
T2	SHAKE DOWN INMATES
T3	SHAKE DOWN VISITORS
T4	ESCORT INMATES
T5	TESTIFY IN COURT
S1	SECONDARY - SUBDUE VIOLENT INMATES
S2	SECONDARY - SHAKE DOWN INMATES
S3	SECONDARY - SHAKE DOWN VISITORS
S4	SECONDARY - ESCORT INMATES
S5	SECONDARY - TESTIFY IN COURT

SYSTEM GROUP COLUMN AGGREGATES

G1	I2	I3					
G2	I4	I5					
G3	I4	I5	I6				
G4	I1	I2	I3				
G5	I4	I5	I6	I7			
G6	I1	I2	I3	I4	I5	I6	I7

ADDATA

INTRODUCTION

PURPOSE

The ADDATA procedure provides the means by which multiple rows or columns may be appended to an existing CODAP80 database. In addition, the user may optionally request that the elements of the rows or columns being appended be relativized to a percentage scale. The ADDATA procedure is particularly useful for adding large amounts of information to a database that was not available when the database was originally created.

FORM

The general form of the ADDATA procedure is as follows:

1) The procedure keyword ADDATA.
2) The keyword ROWS or COLUMNS. This keyword alerts the system that either rows or columns are being added to the database.
3) A group or module designation representing the "length" or number of elements that are contained in the row(s) or column(s) being added.
4) A designation of the number of rows or columns being added and, optionally, those elements of the rows or columns that are to be relativized.
5) A user supplied valid CODAP80 ID (or IDs) to be associated with the added row(s) or column(s).
6) A user supplied FORTRAN format for reading-in the row(s) or columns(s) being added to the database.
7) Options controlling the permanence of the added ID(s), missing value considerations and whether or not the added information is to be printed.

EXAMPLE

```
BEGIN SAMPLEDATA80 EXECUTE.
ADDATA ROWS FOR G6 N=1
     SANDBLASTER 'OPERATE SANDBLASTER'
     FORMAT '(7F1.0)'.
END.
0110010
```

In this example, a single new row named SANDBLASTER is being added to the database. There will be a value of SANDBLASTER for every column associated with the system group ID G6 (I1-I7, see Sample Database). The string OPERATE SANDBLASTER, enclosed in single quotes, is the remark to be associated with the new row. The keyword FORMAT signifies that the row ID SANDBLASTER is to be read with the following format specification that is enclosed in single quotes and parentheses.

OUTPUT FROM PROCEDURE

The result of executing the ADDATA procedure will be new rows or columns optionally added to the database. Specification of the optional keyword LIST .in the syntax of the ADDATA procedure will produce a printed listing of the rows or columns being added.

ADDATA SYNTAX

Refer to the syntax graph of the ADDATA procedure.

ADDATA.

The keyword ADDATA identifies the command.

DATA TYPE DESIGNATION

The keyword ROWS or COLUMNS indicates whether the data being added are conceptual rows or columns of the database.

FOR

The FOR keyword alerts the ADDATA procedure to expect a following group or module ID.

GROUP ID

A group ID is an identified aggregate of database columns. If the preceding data type designation was ROWS, then a group ID must follow the FOR keyword. The group ID may be one previously defined through the use of the SELECT procedure, one of the CODAP80 system cluster groups (as defined at database creation time by the OGROUP routine) or the CODAP80 system group INCUMBENTS. The group ID specification serves to indicate to the ADDATA procedure the database columns for which the new rows are being added. The group ID also serves to indicate the "length" or number of elements the added rows will have.

MODULE ID

The module ID is an identified aggregate of database rows. If the preceding data type designation was COLUMNS, then a module ID must follow the FOR keyword. The module ID may be one previously defined through the use of the SELECT procedure, or may be one of the CODAP80 system modules HVARS, TVARS, TASKS or SVARS. The module ID specification serves to indicate to the ADDATA procedure the database rows for which the new columns are being added. The module ID also serves to indicate the "length" or number of elements the added columns will have.

N
ASSIGNMENT OPERATOR
CONSTANT

The ADDATA procedure requires the user to specify the number of rows or columns that are being appended to the database. For example, were the user to be adding five new rows to the database it would be necessary to appropriately specify the syntax "N=5" to alert the ADDATA procedure of this fact.

REL
CONSTANT LIST

The optional appearance of the keyword REL followed by a constant list indicates to the ADDATA procedure that all or part of the rows or columns to be appended are to be relativized to a 100 point scale. This option allows additional incumbent raw time spent responses to be conveniently converted to percent time spent values.

The constant list following the REL keyword provides the means by which the user can specify which elements of the appended rows or columns are to be relativized. The form of the constant list consists of integer numbers enclosed in parentheses. For example, specification of the constant list "(5, 7, 10-12, 14)" indicates that the fifth, seventh, tenth thru twelfth and fourteenth elements of the appended rows or columns are to be relativized. See Example 1 of ADDATA.

ID

The user has two choices in how ID's can be specified to the ADDATA procedure. Which choice the user picks is determined by whether or not the user wants to individually name each row or column added to the database or to let the procedure append a numeric value to a supplied "seed" ID. See Examples 2 and 3 of ADDATA for illustration.

NOSAVE

Specification of the optional keyword NOSAVE indicates that the added rows or columns will exist on the database only for the duration of the computer run.

REMARK

This is a string of up to 240 characters enclosed in single quotes. The remark will be associated with the added rows or columns. A remark must be associated with the added rows or columns.

LIST

Specification of the optional keyword LIST indicates that a printed listing of the added rows or columns is to be produced.

MISSING
ASSIGNMENT OPERATOR
CONSTANT

Some of the elements of the rows or columns to be added to the database may be missing (as opposed to being zero or blank). To signal the ADDATA procedure that a given value is missing, choose a unique integer constant as the identifier in the missing option. For example, suppose the user was adding a new row to the database, and one of its five elements was missing. By indicating a unique integer constant in the missing option (let's say 99), the ADDATA procedure would then know that any values of 99 that were input as the new row should be set to missing (see ADDATA Example 3).

FORMAT

The FORMAT keyword serves to indicate to the ADDATA procedure that the following string enclosed in single quotes is to be used as the input format for reading-in the values of the rows or columns to be added.

FORMAT SPECIFICATION

The format specification for the ADDATA procedure may be any valid 1966 Ansi Standard FORTRAN format in parentheses, enclosed in single quotes. The format will be used by the ADDATA procedure to read-in the values of the added rows or columns. The place in the input stream of a CODAP80 source language program where the values of the rows or columns to be added are to appear is directly after the terminating END statement (see ADDATA Examples). For an explanation of FORTRAN formats, consult any introductory FORTRAN text.

PERIOD

A period ('.') must end the ADDATA statement.

ADDATA EXAMPLES

EXAMPLE 1

```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT ROWS SYSTEMROWS (H1-H4, T1-T5, S1-S5)
    'ALL SYSTEM ROWS ON DATABASE'.
ADDATA COLUMNS FOR SYSTEMROWS N=1 REL (5-9)
    CASEID_8 'CASEID=8' FORMAT '(F1.0, 2F2.0, 11F1.0)'.
END.
227048796411122
```

The above example is illustrating how the data for an incumbent worker may be added to an already existing database. Raw time spent responses from the incumbent are to be relativized to a 100 point percentage scale.

The SELECT procedure is first being invoked to create a module (named SYSTEMROWS) containing all the system rows on the database (see Sample Database). The ADDATA syntax is requesting that a single new column (as indicated by the N=1 specification) be permanently added to the database, and that it be named CASEID_8. The new column will have an element for every system row on the database (as defined by the created module SYSTEMROWS). The specification REL (5-9) indicates that the fifth thru ninth elements of the column to be added are to be relativized to a 100 point percentage scale (the fifth thru ninth elements of the new column correspond to the task rows of the database). After execution of ADDATA Example 1, the new created column will conceptually reside on the database as follows:

	CASEID_8
H1	2.00
H2	27.00
H3	4.00
H4	8.00
T1	25.93
T2	33.33
T3	22.22
T4	14.81
T5	3.70
S1	1.00
S2	1.00
S3	1.00
S4	2.00
S5	2.00

EXAMPLE 2

```
BEGIN SAMPLEDATA80 EXECUTE.
ADDATA ROWS FOR G6 N=5
     TRACTOR              'OPERATE TRACTOR'
     JACKHAMMER           'OPERATE JACKHAMMER'
     BULLDOZER            'OPERATE BULLDOZER'
     POWERWENCH           'OPERATE POWERWENCH'
     FLAMETHROWER         'OPERATE FLAMETHROWER'
     FORMAT '(7F1.0)'.
END.
1100011
0010100
1100000
1001000
0000001
```

The ADDATA syntax in Example 2 is requesting that 5 new rows be permanently added to the database. The rows will have an element for every incumbent (system) column on the database (as indicated by the system cluster group G6). After execution of ADDATA Example 2, the new created rows will conceptually reside on the database as follows:

	I1	I2	I3	I4	I5	I6	I7
TRACTOR	1.00	1.00	0.00	0.00	0.00	1.00	1.00
JACKHAMMER	0.00	0.00	1.00	0.00	1.00	0.00	0.00
BULLDOZER	1.00	1.00	0.00	0.00	0.00	0.00	0.00
POWERWENCH	1.00	0.00	0.00	1.00	0.00	0.00	0.00
FLAMETHROWER	0.00	0.00	0.00	0.00	0.00	0.00	1.00

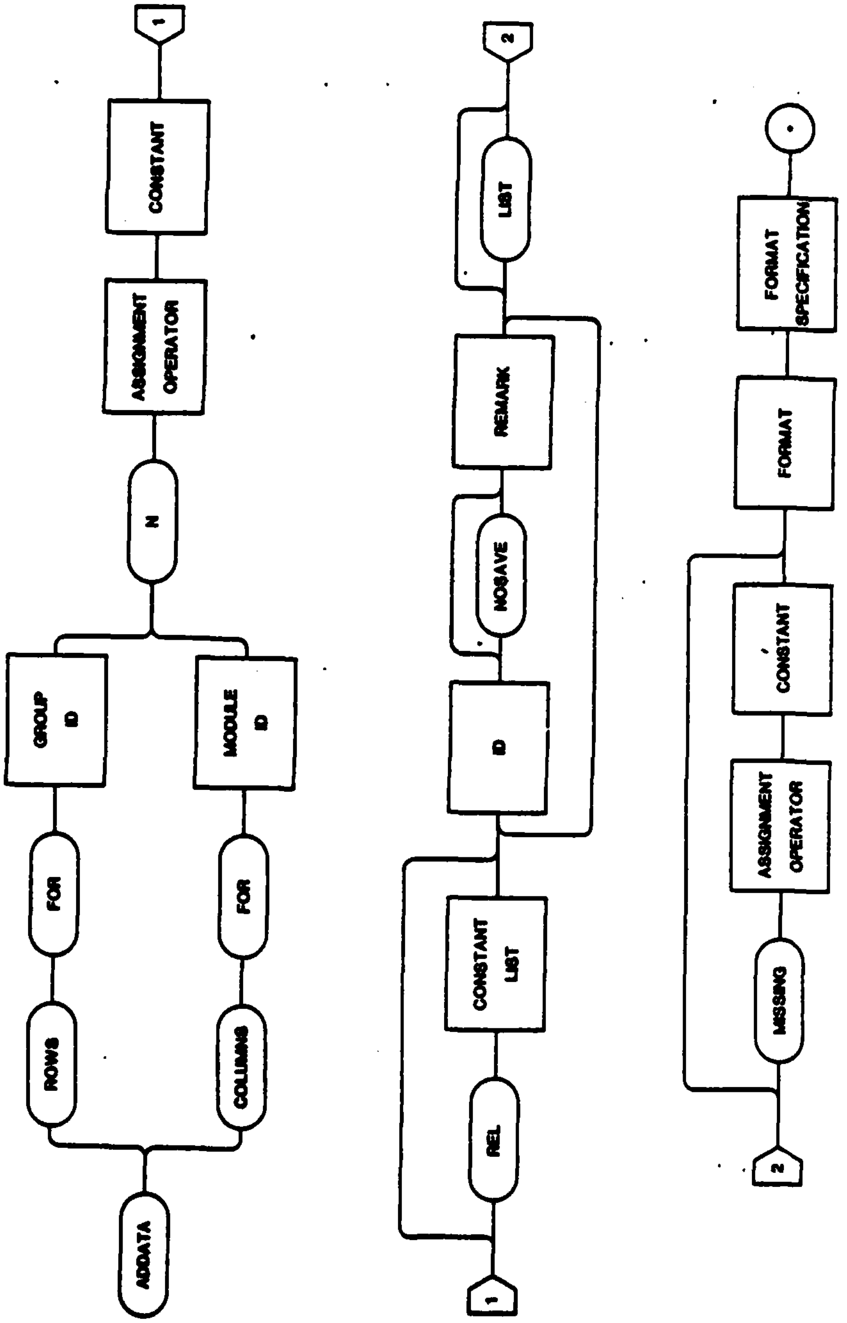
Once the new created rows have been appended to the database, they may be used by other CODAP80 procedures for processing.

EXAMPLE 3

```
BEGIN SAMPLEDATA80 EXECUTE.
ADDATA ROWS FOR G6 N=5
     EQUIPMENT 'OPERATE EQUIPMENT'
     MISSING = 9 FORMAT '(7F1.0)'.
END.
1199911
9919199
1199999
1991999
9999991
```

The ADDATA syntax in Example 3 is requesting basically the same thing as that in Example 2. Namely, that 5 new rows be added to the database. The primary difference between Example 2 and 3 is that, in Example 3, the user has opted not to individually name each of the added rows. Instead, the user has supplied a "seed" ID (EQUIPMENT) that will have a numeric value (beginning with the number 1) appended to it for every row being added.

Specification of the optional syntax MISSING=9 indicates to the ADDATA procedure that any elements of the rows being added that are equal to 9 are to be set to a missing value.


ADDATA
ROWS
FOR
GROUP ID
COLUMNS
FOR
MODULE ID
N
ASSIGNMENT OPERATOR
CONSTANT
1
1
REL
CONSTANT LIST
ID
NOSAVE
REMARK
LIST
2
2
MISSING
ASSIGNMENT OPERATOR
CONSTANT
FORMAT
FORMAT SPECIFICATION
•

AVALUE

INTRODUCTION

PURPOSE

The AVALUE procedure will compute statistics on any specified aggregate of database rows (a module) across one or more specified aggregates of database columns (a group list). In this respect it is similar to the DESCRIBE procedure. The difference between the two though is that unlike the DESCRIBE procedure, which uses the actual values of the row or column being processed to determine the desired statistic, the AVALUE procedure substitutes the values from a specified row for the appropriate values of the rows being processed before determining the desired statistic. The AVALUE procedure is particularly useful in answering questions having to do with determining for each task the average age, income or job title (any database row may be specified) of those incumbents of interest performing the tasks (any module may be specified).

FORM

The general form of the AVALUE command is as follows:

1) The procedure keyword AVALUE.
2) The data type designation ROWS.
3) A description of the aggregate of rows upon which the procedure is to calculate statistics (specified in the form of a module ID).
4) A description of at least one column aggregate (specified in the form of at least one group ID) across which row statistics are to be calculated.
5) A row ID, the values of which will be substituted for the appropriate values of the rows being processed in 3.
6) A new ID. The new ID will have a numeric value, ranging from 1 to the number of group IDs specified in 4, appended to it by the system. The user must be careful not to specify an ID that will conflict with one previously defined in the database. The user must also take care to specify an ID that, when the numeric value is appended to it by the system, is not longer than 12 characters. If only one column aggregate (group ID) is specified in 4, then a numeric value is not appended to the new ID.
7) One of the statistical functions: AVGP, AVGA, STDP, STDA, SUM or N. The function specified defines the type of statistic AVALUE will compute for the values substituted from the row specified in 5.
8) Optionally, the keyword NOSAVE.
9) Descriptive text (a remark) supplied by the user that will be associated with the new column IDs added to the database.

10) Either a period or semicolon. If a period is specified, then the AVALUE command syntax will be finished. If a semicolon is specified, then a new set of syntax described in 6-10 above will follow.

    NOTE: The AVALUE procedure is one of the few procedures in CODAP80 that is not symmetric. AVALUE may only be used to calculate statistics on rows measured across columns.

EXAMPLE

```
AVALUE ROWS TASKS FOR (INCUMBENTS) USING H2
  H2INCAVGP := AVGP NOSAVE
  'AVERAGE AGE FOR INCUMBENTS PERFORMING TASK'.
```

The above AVALUE command syntax will calculate, per task, the average age of the incumbents who are performing the task. An average age (H2) is to be calculated for every task row on the database (as indicated by the system module TASKS) across all incumbent columns (as indicated by the system group INCUMBENTS). The new column ID will be designated H2INCAVGP but will not be permanently saved on the database.

OUTPUT FROM PROCEDURE

Execution of the AVALUE procedure produces no printed output. For every group ID specified in the group list, AVALUE will optionally add a new column to the database.

AVALUE SYNTAX

Refer to the syntax graph of the AVALUE procedure.

AVALUE

The keyword AVALUE identifies the command.

DATA TYPE DESIGNATION

The keyword ROWS indicates that AVALUE is to perform its calculations on rows of the database.

MODULE ID

A module ID is any defined aggregate of database rows. The module ID may be one previously defined through the use of the SELECT procedure, or may be one of the sytem modules HVARS, TVARS, TASKS or SVARS. The module ID specification serves to identify to the AVALUE procedure the database rows upon which statistics are to be calculated.

FOR

The FOR keyword serves to indicate to the AVALUE procedure that the following group list identifies those aggregates of database columns across which calculations are to be performed.

GROUP LIST

A group list is a list of at least one group ID enclosed in parentheses. Created group IDs (such as would be generated by SELECT), system group IDs (such as the keywords INCUMBENTS or INCS) and system group lists (such as G1-G3, as defined by clustering at database creation time) may all appear in a group list. Each group ID specified in the group list represents a different aggregate of database columns across which statistics for a row are to be calculated.

USING

The USING keyword serves to alert the AVALUE procedure that the following row ID is to provide the values to be substituted when calculating the row statistics across column aggregates.

ROW ID

The row ID specification may consist of any existing row on the database. The row ID specified will provide the values to be substituted when calculating row statistics across column aggregates.

ID

A user supplied "seed" ID. Appended to this ID will be a numeric value, ranging from 1 to the number of group IDs specified in the group list (unless, of course, only a single group ID appeared in the group list). Because AVALUE is not symmetric, all new IDs generated by this procedure pertain to added database columns.

ASSIGNMENT OPERATOR

Either of the symbols '=' or ':='. Either of these symbols may be used to separate the seed ID from the statistical function that follows.

STATISTICAL FUNCTIONS

The statistical function specified defines the type of statistical operation performed across columns by AVALUE on the substituted row values. The six acceptable statistical function keywords are as follows:

*AVGP	-	Average, excluding missing values.
AVGA	-	Average, including missing values.
*STDP	-	Standard deviation, excluding missing values.
STDA	-	Standard deviation, including missing values.
SUM	-	Sum of non-missing values.
*N	-	Number of non-missing values.

*If a calculation is being performed on task rows across columns, zeros are interpreted as missing.

NOSAVE

Specification of the optional keyword NOSAVE indicates that any new columns generated through the execution of the AVALUE procedure are <u>not</u> to be permanently saved for future reference.

REMARK

This is a string of up to 240 characters, enclosed in single quotes. The remark will be associated with the new column IDs generated. A remark <u>must</u> be associated with the new IDs.

PERIOD OR SEMICOLON

A period ('.') must end the syntax of the AVALUE procedure. If the user desires to calculate more than one statistic on the same database subset, the command syntax may be terminated with a semicolon, followed by the specification of a new ID, statistical function and a remark (see AVALUE example 1).

AVALUE EXAMPLES

EXAMPLE 1

```
AVALUE ROWS TASKS FOR (G6) USING H2
     AVGPAGE := AVGP
       'AVERAGE AGE (AVGP), G6';
     STDPAGE := STDP
       'STD AGE (STDP), G6'.
```

The above AVALUE command syntax will calculate, for each task row on the database (as designated by the system module TASKS), the average and standard deviation (AVGP and STDP, missing values excluded) of age (H2) for those incumbents performing the task. All incumbent columns of the database (I1-I7) will be included in the calculations owing to the specification of the system cluster group G6. Execution of the above syntax will result in two created columns, each 5 elements long (one for each task) being added to the database. The created column AVGPAGE will contain the average age of the incumbents performing the tasks and column STDPAGE will contain the age standard deviations. Note the use of the semicolon in the command's syntax.

The command syntax AVALUE ROWS TASKS FOR (G6) USING H2 defines the following data subsets of the Sample Database that will go into the computations:

	I1	I2	I3	I4	I5	I6	I7
T1	64	11	0	11	24	36	0
T2	9	11	0	44	24	64	43
T3	9	22	20	0	18	0	57
T4	18	56	50	22	0	0	0
T5	0	0	30	22	35	0	0
H2	19	23	.	41	27	53	.

The values that will be computed for the five elements (one per task) of the created column AVGPAGE are as follows:

```
AVGPAGE (1) = (19+23+41+27+53)/5 = 32.6
AVGPAGE (2) = (19+23+41+27+53)/5 = 32.6
AVGPAGE (3) = (19+23+27)/3       = 23.0
AVGPAGE (4) = (19+23+41)/3       = 27.7
AVGPAGE (5) = (41+27)/2          = 34.0
```

The values that will be computed for the five elements (one per task) of the created column STDPAGE are as follows:

$$\text{STDPAGE (1)} = (((19^2+23^2+41^2+27^2+53^2)-((19+23+41+27+53)^2/5))/4)^{.5} = 14.1$$

$$\text{STDPAGE (2)} = (((19^2+23^2+41^2+27^2+53^2)-((19+23+41+27+53)^2/5))/4)^{.5} = 14.1$$

$$\text{STDPAGE (3)} = (((19^2+23^2+27^2)-((19+23+27)^2/3))/2)^{.5} \qquad = \ 4.0$$

$$\text{STDPAGE (4)} = (((19^2+23^2+41^2)-((19+23+41)^2/3))/2)^{.5} \qquad = 11.7$$

$$\text{STDPAGE (5)} = (((41^2+27^2)-((41+27)^2/2))/1) \qquad = \ 9.9$$

The fact that the keyword NOSAVE did not appear in the syntax of AVALUE example 1 indicates that the two columns (AVEPAGE and STDPAGE) are to be saved permanently on the database. For an illustration of a report displaying the created columns produced above, the reader is referred to PRINT example 1.

EXAMPLE 2

```
AVALUE ROWS TASKS FOR (G4, G5) USING H2
     AVGAAGE := AVGA
     'AVERAGE AGE (AVGA), G4 AND G5'.
```

The above AVALUE command syntax will calculate, for each task row on the database (as designated by the CODAP80 system module TASKS), the average (AVGA, missing values included) of age (H2) for those incumbent columns identified by the CODAP80 system cluster group G4 (I1-I3), and then again for the incumbent columns in system cluster group G5 (I4-I7). Execution of the above syntax will result in two created columns, each 5 elements long (one for each task) being added to the database permanently. The two created columns will be named AVGAAGE1 and AVGAAGE2 (the numerals being appended to the seed ID as a function of the number of group IDs appearing in the group list). Note closely the difference in the computational process applied between the statistical functions AVGP (used in example 1) and AVGA (used in the present example).

The command syntax AVALUE ROWS TASKS FOR (G4, G5) USING H2 defines the following data subsets of the Sample Database that will go into the computations:

	G4			G5			
	I1	I2	I3	I4	I5	I6	I7
T1	64	11	0	11	24	36	0
T2	9	11	0	44	24	64	43
T3	9	22	20	0	18	0	57
T4	18	56	50	22	0	0	0
T5	0	0	30	22	35	0	0
H2	19	23	.	41	27	53	.

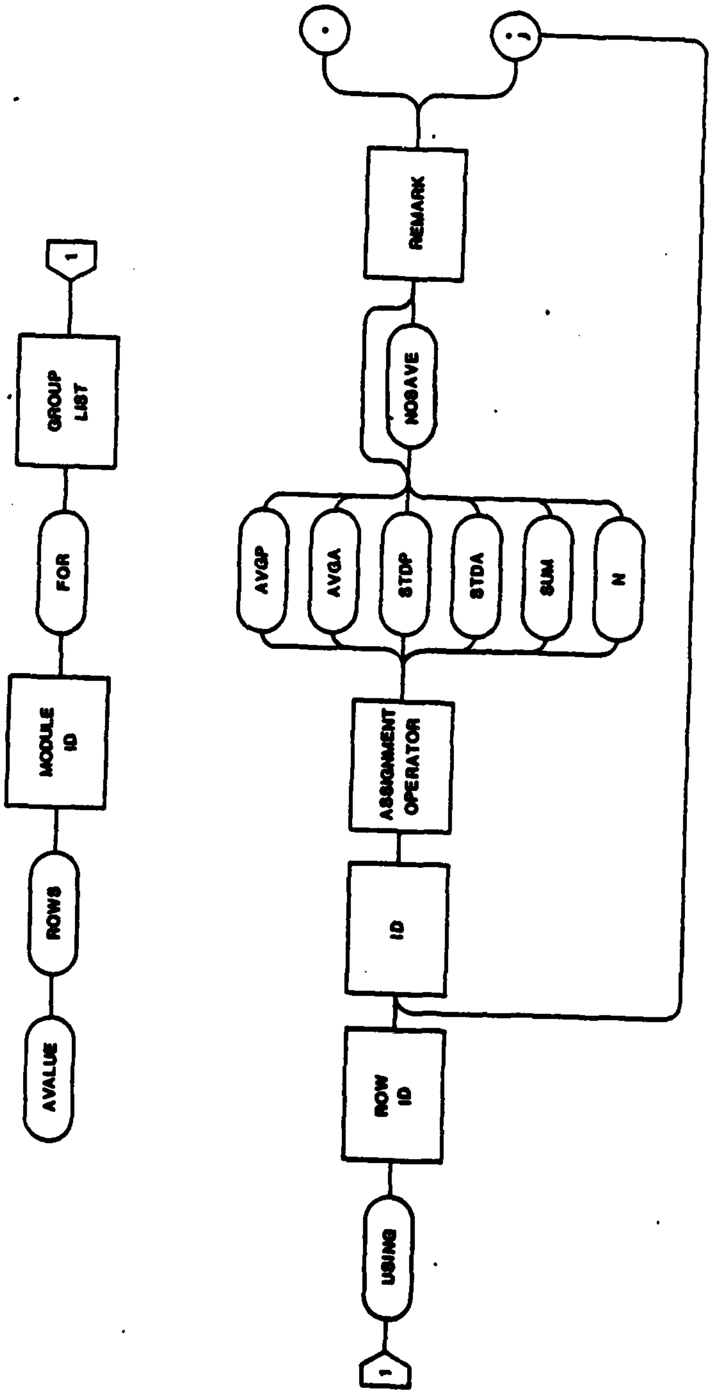
The values that will be computed for the five elements (one per task) of the created column AVGAAGE1 are as follows:

```
AVGAAGE1(1) = (19+23)/2 = 21.00
AVGAAGE1(2) = (19+23)/2 = 21.00
AVGAAGE1(3) = (19+23)/3 = 14.00
AVGAAGE1(4) = (19+23)/3 = 14.00
AVGAAGE1(5) =           =  .
```

The values that will be computed for the five elements (one per task) of the created column AVGAAGE2 are as follows:

```
AVGAAGE2(1) = (41+27+53)/3 = 40.33
AVGAAGE2(2) = (41+27+53)/4 = 30.25
AVGAAGE2(3) = (27)/2       = 13.50
AVGAAGE2(4) = (41)/1       = 41.00
AVGAAGE2(5) = (41+27)/2    = 34.00
```

AVALUE


AVALUE
ROWS
MODULE ID
FOR
GROUP LIST
1
1
USING
ROW ID
ID
ASSIGNMENT OPERATOR
AVGP
AVGA
STDP
STDA
SUM
N
NOSAVE
REMARK
.
;

BEGIN

INTRODUCTION

PURPOSE

The purpose of the BEGIN command is to delineate the beginning of a CODAP80 source language program, to inform the CODAP80 interpreter whether or not the execution phase is to be entered following syntax analysis and to inform the CODAP80 interpreter of the study ID of the database to be processed.

FORM

The general form of the BEGIN command is as follows:

1) The procedure keyword BEGIN.
2) An indication of the database study ID.
3) Optionally, the keyword EXECUTE.

EXAMPLE

```
BEGIN SAMPLEDATA80 EXECUTE.
```

The above BEGIN command syntax is alerting CODAP80 that a source language program follows. The database to be processed has the ID SAMPLEDATA80. The appearance of the keyword EXECUTE indicates that following syntax analysis, if no syntax errors were found, the execution phase of the CODAP80 interpreter is to be entered.

OUTPUT FROM PROCEDURE

Execution of the BEGIN procedure produces no printed output. The BEGIN procedure is not a procedure in the sense that, say AVALUE or DESCRIBE are procedures. BEGIN performs no calculations on values in the database. BEGIN serves only to alert CODAP80 that a source language program is being submitted.

BEGIN SYNTAX

Refer to the syntax graph of the BEGIN procedure.

BEGIN

The keyword BEGIN identifies the command.

STUDY ID

During the database creation phase of an occupational study a study ID was assigned by the user. The assignment of a study ID takes place following successful execution of the INPSTD database creation routine. The study ID supplied by the user following the BEGIN command keyword is checked against that stored on the database and, if they match, processing continues. If the study ID stored on the database and the study ID supplied by the user in the BEGIN command do not match, an error is indicated and processing immediately ceases.

EXECUTE

Specification of the optional keyword EXECUTE indicates to CODAP80 that the execution phase of the interpreter is to be entered following analysis of the source language statements making up the CODAP80 program. The execution phase will only be entered if no syntax errors are found. If the EXECUTE keyword is omitted, then processing will automatically cease following syntax analysis.

PERIOD

A period ('.') must terminate the BEGIN command.

BEGIN EXAMPLES

EXAMPLE 1

```
BEGIN SAMPLEDATA80.
DESCRIBE ROWS TASKS FOR (G6)
  NEWCOLUMN := AVGA 'A NEW COLUMN'.
END.
```

The above CODAP80 source language statements would be analysed for syntactical errors only. The fact that the optional keyword EXECUTE is not included in the BEGIN statement will prevent the execution phase of the CODAP80 interpreter from being entered.

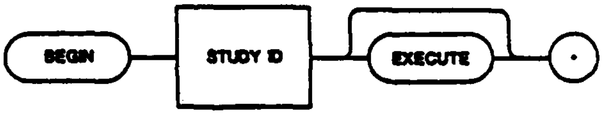
EXAMPLE 2

```
BEGIN SAMPLEDAT80 EXECUTE.
```

In the above BEGIN statement the study ID has been incorrectly specified (SAMPLEDATA80 is the correct study ID). This would result in the following error message:

```
********************************************************************************
STUDY ID - SAMPLEDAT80 IS INVALID.  THE INTERPRETER HAS TO BE
STOPPED FOR SECURITY AND PROTECTION.  PLEASE CONSULT USER
MANUAL OR CHANGE TO THE CORRECT STUDY ID.
********************************************************************************
```

BEGIN



CLUSTER

INTRODUCTION

PURPOSE

The CLUSTER procedure will perform hierarchical clustering (based on Ward, 1963) either on any set of columns (a group) of the database measured across any set of rows, or on any set of rows (a module) measured across any set of columns. In addition, the user will have the option of requesting any one of four techniques for calculating measures of similarity between columns or rows (see Appendix B for overlap similarity formulae).

FORM

The general form of the CLUSTER procedure is as follows:

1) The procedure keyword CLUSTER.
2) A designation of the columns or rows of the database to be clustered.
3) A designation of which measure of similarity between columns or rows is to be used by the procedure.
4) A minimum membership designation for the diagram display.
5) A user assigned ID to be associated with the clustered row or column hierarchical sequence number (HSN).

EXAMPLE

```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT COLUMNS MALES (H1=1) 'INCUMBENTS OF THE MALE SEX'.
CLUSTER COLUMNS MALES FOR TASKS OVL MAXIMIZE
     MALEHSN
     'HSN NUMBER FROM CLUSTERING MALE INCUMBENTS'
     MINMEM=2
     HEADING='CLUSTERING MALE INCUMBENTS'.
END.
```

In the above example, the male incumbents are first being selected into group MALES. The CLUSTER command is then requesting that the elements of this group be clustered, and that the overlap calculated between the elements be measured across all the task rows on the database. The overlap algorithm is to be absolute overlap. The HSN values generated for each column being clustered will be assigned the ID MALEHSN. The diagram produced will have a starting minimum group membership of 2.

OUTPUT FROM PROCEDURE

Output from the CLUSTER procedure will consist of a group membership report detailing the clustering process and a diagram report detailing the clustering process pictorially. HSN values generated for the rows or columns clustered may be added to the database for further processing by other procedures.

CLUSTER SYNTAX

Refer to the syntax graph of the CLUSTER procedure.

CLUSTER

The keyword CLUSTER identifies the command.

DATA TYPE DESIGNATION

The keyword COLUMNS indicates that the CLUSTER procedure is to cluster columns on the database. The keyword ROWS indicates that rows on the database are to be clustered.

GROUP OR MODULE ID

If the data type designation is COLUMNS, then a group ID must follow. If ROWS is designated, a module ID must follow. The group or module ID specifies which columns or rows of the database are to be clustered.

FOR

The keyword FOR alerts the procedure that the following data designation represents the values of the database across which similarity between the rows or columns being clustered is to be calculated.

MODULE OR GROUP ID

The module or group ID following the FOR keyword defines the values across which similarity between the rows or columns being clustered is to be calculated. If a group ID occurs before the FOR keyword, then a module ID must follow. Conversely, if a module ID occurs before the FOR keyword, then a group ID must follow.

OVERLAP

OVERLAP (or OVL) is one of the options available for calculating similarity between columns or rows. The similarity coefficient calculated will be the sum of the absolute overlaps between columns or rows.

DSQUARE

DSQUARE is one of the options available for calculating similarity between columns or rows. With this option, the similarity coefficient calculated will be the sum of the squared deviations between columns or rows.

D

D is one of the options available for calculating similarity between columns or rows. With this option, the similarity coefficient calculated will be the sum of the deviations between columns or rows.

BINARY

BINARY is one of the options available for calculating similarity between columns or rows. The similarity coefficient calculated will be a function of the response - nonresponse profile agreement between columns or rows on the database.

MAXIMIZE

Specifying the MAXIMIZE keyword instructs the system to cluster most similar columns or rows first.

ID

Any valid CODAP80 ID supplied by the user to be associated with the HSN values generated for the clustered rows or columns.

REMARK

This is a string of up to 240 characters enclosed in single quotes. The remark will be associated with the user specified ID. A remark must be associated with the added row or column.

MINMEM ASSIGNMENT OPERATOR CONSTANT

The user must specify the minimum membership for the diagram starter groups. A valid example would be "MINMEM=10". A value less than 2 will produce an error.

HEADING

The keyword HEADING indicates that the following text string enclosed in quotes is to be used as a report title.

CHARACTER STRING

Up to 10 lines of 131 characters each may comprise the title character string.

PERIOD

A period ('.') must end the CLUSTER statement.

CLUSTER EXAMPLES

EXAMPLE 1

```
 BEGIN SAMPLEDATA80 EXECUTE.
 SELECT ROWS EQUIPMENT (TRACTOR JACKHAMMER BULLDOZER
     POWERWENCH) 'EQUIPMENT OPERATED'.
 CLUSTER COLUMNS INCUMBENTS FOR EQUIPMENT
     BINARY MAXIMIZE
     INCHSN 'HSN—CLUSTERING INCUMBENTS FOR EQUIP—BINARY'
     MINMEM=2
     HEADING='CLUSTERING INCUMBENTS FOR EQUIPMENT'.
 END.
```

The CODAP80 syntax in example 1 of CLUSTER is requesting that a module of created database rows (see ADDATA example 2) be selected. Following that, the CLUSTER command will perform a hierarchical clustering on the incumbent columns of the database, with the similarity between incumbents being a function of their performance-nonperformance profile (BINARY) on the equipment rows identified by the created module EQUIPMENT. The diagram produced will have a minimum starter group membership of 2. Had a "PRTVAR" report been desired, the user need only to specify a PRINT command sorting on the ID INCHSN.

EXAMPLE 2

```
 BEGIN SAMPLEDATA80 EXECUTE.
 CLUSTER ROWS TASKS FOR INCUMBENTS
     BINARY MAXIMIZE
     TASKHSN 'TASK HSN — BINARY OVERLAP'
     MINMEM=2
     HEADING=
'CLUSTERING TASKS FOR SYSTEM COLUMNS — BINARY OVERLAP'.
 END.
```

The above syntax is requesting that all the task rows of the database be clustered with the overlap between task rows being measured across all the incumbents as a function of their performance-nonperformance response profile (binary overlap). A new column of HSN values (named TASKHSN) will be generated for the clustered task rows. The diagram produced will have a minimum starter group membership of 2.

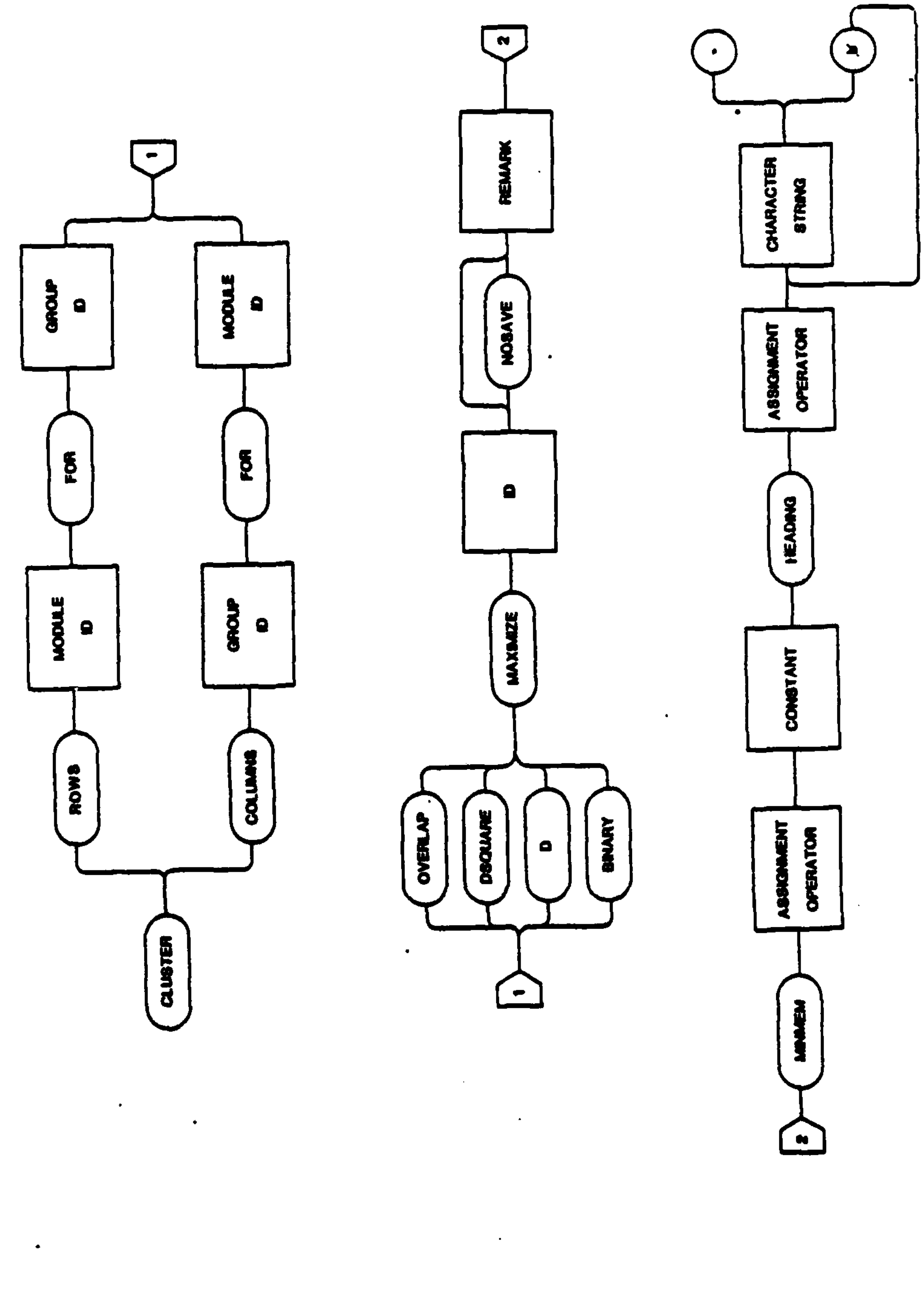
EXAMPLE 2
PRINTED OUTPUT

```
                                             STUDY ID - SAMPLEDATA80
                                    CLUSTERING TASKS FOR SYSTEM COLUMNS -- BINARY OVERLAP

                   RESULTANT GROUP                   FORMED BY                             COMBINING GROUPS
          GROUP    NO.    SEQUENCE        AVERAGE                       NUM     AT      SEQUENCE                NUM     AT      SEQUENCE
 STAGE    IDENT   MBERS   FROM   TO   BETWEEN     WITHIN      IDENT    MBRS   STAGE    FROM   TO      IDENT    MBRS   STAGE    FROM   TO
     1        1       2      1    2   83.3333     83.3333         1       1       0       0    0          2       1       0       0    0
     2        3       2      3    4   50.0000     50.0000         3       1       0       0    0          4       1       0       0    0
     3        1       4      1    4   48.2143     54.3651         1       2       1       1    2          3       2       2       3    4
     4        1       5      1    5   33.8095     46.1428         1       4       3       1    4          5       1       0       0    0
```

EXAMPLE 2
PRINTED OUTPUT (continued)

```
                              STUDY ID - SAMPLEDATA80
                  CLUSTERING TASKS FOR SYSTEM COLUMNS -- BINARY OVERLAP

                                  MINMEM=    2

  1 2         2 2
  1-2         3-4
83.  83.    50.  50.
   |           |
   |           |
   |           |
  3 4          |
  1-4  --------
48.  54.
   |
   |
   |
  4 5
  1-5
33.  46.
```

CLUSTER


CLUSTER
ROWS
MODULE ID
FOR
GROUP ID
COLUMNS
GROUP ID
FOR
MODULE ID
1
1
OVERLAP
DSQUARE
D
BINARY
MAXIMIZE
ID
NOSAVE
REMARK
2
2
MINMEM
ASSIGNMENT OPERATOR
CONSTANT
HEADING
ASSIGNMENT OPERATOR
CHARACTER STRING
,

COPY

INTRODUCTION

PURPOSE

The COPY command copies any number of specified rows or columns on the database to a punch or tape/disk output file.

FORM

The general form of the COPY command is as follows:

1) The procedure keyword COPY.
2) The keyword ROWS or COLUMNS - this keyword alerts the system that either rows or columns of the database are to be copied.
3) A description of which rows or columns on the database are to be copied (the MROWLT or GCOLST designation).
4) A group or module designation representing the "length" or the number of elements that are contained in the row(s) or column(s) that is being copied.
5) An indication of whether or not an identifier is to be punched in columns 1-12 of the output punch record.
6) An indication of whether the rows(s) or columns(s) are to be copied to a punch or tape/disk output file.

EXAMPLE

```
COPY ROWS (HVARS) FOR G5 'HVARS -- G5' CARD.
```

The above COPY command syntax is requesting that the rows identified by the system module HVARS (H1-H4) be copied to a punch file. Only 4 elements of each row are to be copied due to the G5 system group identifier appearing after the FOR keyword (G5 is a system group generated by clustering at database creation time. G5 contains 4 members: I4-I7). The character string HVARS -- G5 is to be punched in columns 1-12 of every 80 character output record.

OUTPUT FROM PROCEDURE

Output from the COPY procedure consists of 80 character records (card images) sent to the punch or tape/disk logical unit. The number of records output by COPY is determined by the number and length of rows or columns being copied.

COPY SYNTAX

Refer to the syntax graph of the COPY procedure.

COPY

The keyword COPY identifies the command.

DATA TYPE DESIGNATION

The keyword ROWS or COLUMNS indicates whether rows or columns of the the database are to be copied.

MODULE ROW LIST

A Module Row List (MROWLT) is a list of at least one module or row ID enclosed in parentheses. Lists of module IDs, system row lists and lists of row IDs may all occur together in a MROWLT. If the data type designation following the COPY command keyword is ROWS, then a MROWLT must follow. The MROWLT serves to indicate to the COPY procedure which rows of the database are to be copied.

GROUP COLUMN LIST

A Group Column List (GCOLST) is a list of at least one group or column ID enclosed in parentheses. Lists of group IDs, system column lists, system group lists and lists of column IDs may all occur together in a GCOLST. If the data type designation following the COPY command keyword is COLUMNS, then a GCOLST must follow. The GCOLST serves to indicate to the COPY procedure which columns of the database are to be copied.

FOR

The FOR keyword alerts the COPY procedure to expect a following group or module ID.

GROUP ID

A group ID is an identified aggregate of database columns. A group ID following the FOR keyword indicates the number of elements comprising, or length of, the row or rows being copied. If the preceding data type designation was ROWS, then a group ID must follow the FOR keyword.

MODULE ID

A module ID is an identified aggregate of database rows. A module ID following the FOR keyword indicates the number of elements comprising, or length of, the column or columns being copied. If the preceding data type designation was COLUMNS, then a module ID must follow the FOR keyword.

CHARACTER STRING

This is a string of up to 12 characters, enclosed in single quotes. The characters appearing between the single quotes will be punched in columns 1-12 of every record output by the COPY procedure. If a character string is not specified, then the system defaults to placing blanks in the first 12 columns of the output record.

CARD OR TAPE

If the keyword CARD appears in COPY's syntax, the specified row(s) or column(s) are to be copied to a punch logical unit. If TAPE appears, the copied row(s) or column(s) will be sent to a tape or disk logical unit.

PERIOD

A period ('.') must end the COPY statement.

COPY EXAMPLES

EXAMPLE 1

COPY ROWS (H2-H4) FOR INCUMBENTS 'H2-H4 INCS.' CARD..

The above COPY statement syntax is requesting that each of the rows specified in the MROWLT (H2-H4), each of length seven (one element in the row for every incumbent in the database), be copied to a punch file. The identifier H2-H4 INCS. is to be punched in columns 1-12 of each record output from the execution of the above COPY procedure syntax.

Referring to the Sample Database, the data to be copied consists of the rows H2, H3 and H4, each row having one element for each incumbent:

	I1	I2	I3	I4	I5	I6	I7
H2	19	23	.	41	27	53	.
H3	1	2	11	19	3	30	16
H4	1	5	7	2	4	6	3

Referring to example 1 of COPY's punched output, it can be seen that the three rows have been copied to a punch file in the following configuration:

The *first* record (or card image) appearing with the punched deck of rows will always be a header record. The header record will have the following information punched on it:

CARD COLUMNS	INFORMATION PUNCHED
1 - 12	The text of the character string that was supplied in the syntax of the COPY command; H2-H4 INCS. in this example.
16 - 28	The text string HEADER RECORD. This text string serves only to help differentiate the header record from the data records.
32 - 43	The study ID. In this example, SAMPLEDATA80.
46 - 50	Number of rows/columns copied. In this example, 3.
51 - 55	Number of elements (or length) of each row/column copied. In this example, 7.
56 - 60	Number of records (or card images) output for each row/column copied. In this example, 2.
61 - 65	Total number of records (or card images) output in copying the rows/columns. In this example, 6.

71 - 71	The text R or C, depending on whether it was rows or columns that were copied. In this example, R.

The records following the header record contain the rows that were copied. The data records after the header record have the following configuration:

CARD COLUMNS	INFORMATION PUNCHED
1 - 12	The text of the character string that was supplied in the syntax of the COPY command; H2-H4 INCS. in this example.
13 - 72	This field will contain up to 5 values of a row/column, punched in E format E12.5. Missing values will appear as -0.10000E+51. If the row/column is longer than 5 elements, it will be continued on to the next record in columns 13-72.
73 - 73	The alphabetic character R or C, depending on if the values being punched on the record are rows or columns.
74 - 76	Row/Column sequence number. In example 1, row H2 was the first row copied. It required two data records of output to copy row H2. Since H2 was the first row copied, the number 1 will appear in columns 74-76 of the first two data records output.
77 - 80	Row/Column record sequence number. In example 1, row H2 required two output records to copy its full length (7). To indicate this, a 1 and than a 2 are punched as sequence numbers in columns 77-80 of output data records 1 and 2.

EXAMPLE 1
COPIED OUTPUT

```
    CARD
--- COLUMNS
|
|
|            1         2         3         4         5         6         7         8
--> 12345678901234567890123456789012345678901234567890123456789012345678901234567890

    H2-H4 INCS.     HEADER RECORD   SAMPLEDATA80     3    7    2    6     R
    H2-H4 INCS.  0.19000E+02 0.23000E+02-0.10000E+51 0.41000E+02 0.27000E+02R  1   1
    H2-H4 INCS.  0.53000E+02-0.10000E+51                                    R  1   2
    H2-H4 INCS.  0.10000E+01 0.20000E+01 0.11000E+02 0.19000E+02 0.30000E+01R  2   1
    H2-H4 INCS.  0.20000E+02 0.16000E+02                                    R  2   2
    H2-H4 INCS.  0.10000E+01 0.50000E+01 0.70000E+01 0.20000E+01 0.40000E+01R  3   1
    H2-H4 INCS.  0.60000E+01 0.30000E+01                                    R  3   2
```

EXAMPLE 2

```
BEGIN SAMPLEDATA80 EXECUTE.
COPY COLUMNS (G2) FOR TASKS 'G2 TASKS' TAPE.
END.
```

The above CODAP80 syntax represents a complete run stream. The COPY statement is requesting that the database columns identified by the system cluster group ID G2 (which are, referring to the Sample Database, I4 and I5) be copied to a tape/disk file. Each of the copied columns will be five elements long (one for each row identified by the system reserved keyword TASKS). The character string G2 TASKS will be placed in columns 1-12 of each column output record.

EXAMPLE 2
COPIED OUTPUT

```
     CARD
 --- COLUMNS
|
|             1         2         3         4         5         6         7         8
 --> 12345678901234567890123456789012345678901234567890123456789012345678901234567890

     G2 TASKS      HEADER RECORD   SAMPLEDATA80       2    5    1    2    C
     G2 TASKS     0.11111E+02 0.44444E+02-0.10000E+51 0.22222E+02 0.22222E+02C  1   1
     G2 TASKS     0.23529E+02 0.23529E+02 0.17647E+02-0.10000E+51 0.35294E+02C  2   1
```

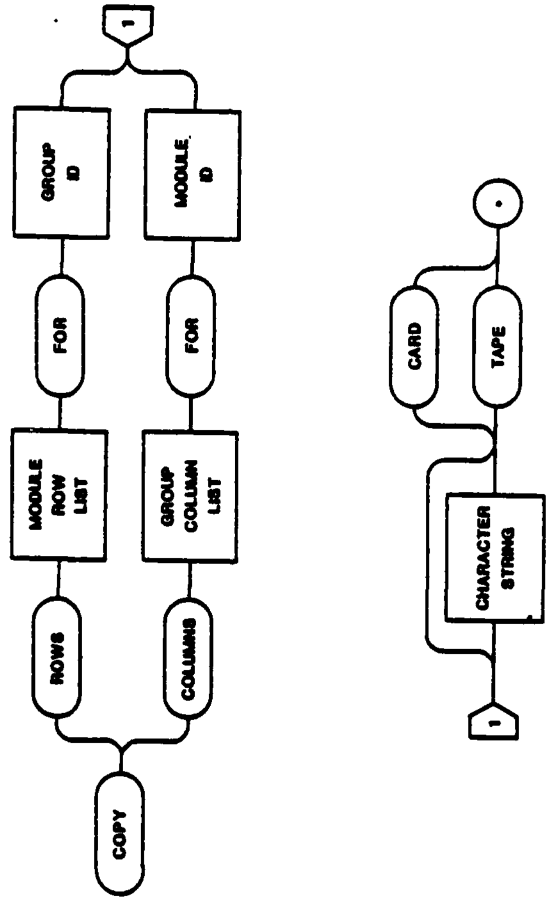
EXAMPLE 3

```
BEGIN SAMPLEDATA80 EXECUTE.
COPY ROWS (HVARS S1) FOR G4 TAPE.
END.
```

The above COPY syntax is requesting that the rows identified in the MROWLT (HVARS, which consists of the rows H1-H4, plus the row S1) be copied to a tape/disk file. Each of these rows will be G4 elements long (G4 is a system cluster group identifying the database columns I1, I2 and I3). Since a character string was not specified, columns 1-12 of each row output record will be blank.

EXAMPLE 3
COPIED OUTPUT

```
     CARD
 --- COLUMNS
|
|             1         2         3         4         5         6         7         8
 --> 12345678901234567890123456789012345678901234567890123456789012345678901234567890

                   HEADER RECORD   SAMPLEDATA80       5    3    1    5    R
                  0.20000E+01 0.10000E+01 0.20000E+01                      R  1   1
                  0.19000E+02 0.23000E+02-0.10000E+51                      R  2   1
                  0.10000E+01 0.20000E+01 0.11000E+02                      R  3   1
                  0.10000E+01 0.50000E+01 0.70000E+01                      R  4   1
                 -0.10000E+51-0.10000E+51-0.10000E+51                      R  5   1
```


COPY
ROWS
COLUMNS
MODULE ROW LIST
GROUP COLUMN LIST
FOR
FOR
GROUP ID
MODULE ID
1
1
CHARACTER STRING
CARD
TAPE
*

CORR

INTRODUCTION

PURPOSE

The CORR command calculates Pearson product-moment correlations of specified rows or columns of the database.

FORM

The general form of the CORR command is as follows:

1) The procedure keyword CORR.
2) The keyword ROWS or COLUMNS - this keyword alerts the system that either rows or columns of the database are to be correlated.
3) A description of which rows or columns of the database are to be correlated.
4) A group or module designation representing the number of observations in the correlation.
5) A description of what is to be printed as a title at the top of the produced report.

EXAMPLE

```
CORR ROWS (H1, H2, H3) FOR G6
     HEADING:='EXAMPLE CORRELATION SETUP'.
```

The above CORR command syntax is requesting that the rows H1, H2, H3 be correlated across the columns identified by the cluster group ID G6 (which is, referring to the sample database, all of the incumbents).

OUTPUT FROM PROCEDURE

Printed output generated from execution of the CORR command consists of a matrix of Pearson Product-Moment correlations of those rows or columns specified in the CORR command's syntax. Printed below each correlation will be the number of pairs of observations that went into the correlation. If the keyword NOREMARKS is not specified, the remarks associated with the rows or columns being correlated will be printed at the beginning of the output.

CORR SYNTAX

Refer to the syntax graph of the CORR procedure.

CORR

The keyword CORR identifies the command.

DATA TYPE DESIGNATION

The keyword ROWS or COLUMNS indicates whether rows or columns of the database are to be correlated.

MODULE ROW LIST

A Module Row List (MROWLT) is a list of at least one module or row ID enclosed in parentheses. Lists of module IDs, system row lists and lists of row IDs may all occur together in a MROWLT. If the data type designation following the CORR command keyword is ROWS, then a MROWLT must follow. The MROWLT serves to indicate to the CORR procedure which rows of the database are to be correlated.

GROUP COLUMN LIST

A Group Column List (GCOLST) is a list of at least one group or column ID enclosed in parentheses. Lists of group IDs, system column lists and lists of column IDs may all occur together in a GCOLST. If the data type designation following the CORR command keyword is COLUMNS, then a GCOLST must follow. The GCOLST serves to indicate to the CORR procedure which columns of the database are to be correlated.

FOR

The FOR keyword alerts the CORR procedure to expect a following group or module ID.

GROUP ID

A group ID is an identified aggregate of database columns. A group ID following the FOR keyword indicates the columns of the database the rows are to be correlated across. If the preceeding data type designation was ROWS, than a group ID must follow the FOR keyword.

MODULE ID

A module ID is an identified aggregate of database rows. A module ID following the FOR keyword indicates the rows of the database the columns are to be correlated across. If the preceeding data type designation was COLUMNS, then a module ID must follow the FOR keyword.

PEARSON

The keyword PEARSON indicates that Pearson product moment correlations are to be calculated. This keyword is optional and need not be specified.

NOREMARKS

Specifying NOREMARKS indicates that the comments associated with the rows or columns that are being correlated are not to be printed at the beginning of the CORR procedure's output. If this keyword is omitted, variable comments will be printed.

HEADING

The keyword HEADING serves to indicate that the following string is to be used as a report title.

ASSIGNMENT OPERATOR

Either the symbols '=' or ':='. Either of these symbols may be used to separate the HEADING keyword from the title character string.

CHARACTER STRING

Up to 10 lines of 131 characters each may comprise the character string(s) that make up the report title of the CORR command. Each title line of up to 131 characters is enclosed in single quotes, with the beginning of a new title line indicated by a blank and another line enclosed in single quotes.

PERIOD

A period ('.') must end the CORR statement.

CORR EXAMPLES

EXAMPLE 1

```
CORR ROWS (H1,H2,H3) FOR INCUMBENTS
     HEADING:='EXAMPLE 1'
     'CORRELATION MATRIX OF ROW VARIABLES H1, H2, & H3'
     'ACROSS ALL INCUMBENTS'.
```

The above CORR statement syntax is requesting that the rows H1, H2 and H3 be correlated across all incumbents in the database (the group ID INCUMBENTS is a CODAP80 reserved keyword).

EXAMPLE 1
PRINTED OUTPUT

```
                                                                  PAGE - 1


                        STUDY ID - SAMPLEDATA80
                               EXAMPLE 1
              CORRELATION MATRIX OF ROW VARIABLES H1, H2 & H3
                         ACROSS ALL INCUMBENTS

                   ROW/COLUMN ID          ROW/COLUMN REMARK
                   -------------          -----------------

                   H - 1                  SEX
                   H - 2                  AGE
                   H - 3                  YEAR ON JOB

**************************************************************************

                                                                  PAGE - 2


                        STUDY ID - SAMPLEDATA80
              CORRELATION COEFFICIENTS/NUMBER OF OBSERVATIONS
                               EXAMPLE 1
              CORRELATION MATRIX OF ROW VARIABLES H1, H2 & H3
                         ACROSS ALL INCUMBENTS


                                H - 1        H - 2        H - 3
                              --------     --------     --------

                  H - 1        1.00000     -0.53921     -0.36364
                                     7            5            7

                  H - 2       -0.53921      1.00000      0.98915
                                     5            5            5

                  H - 3       -0.36364     -0.98915      1.00000
                                     7            5            7
```

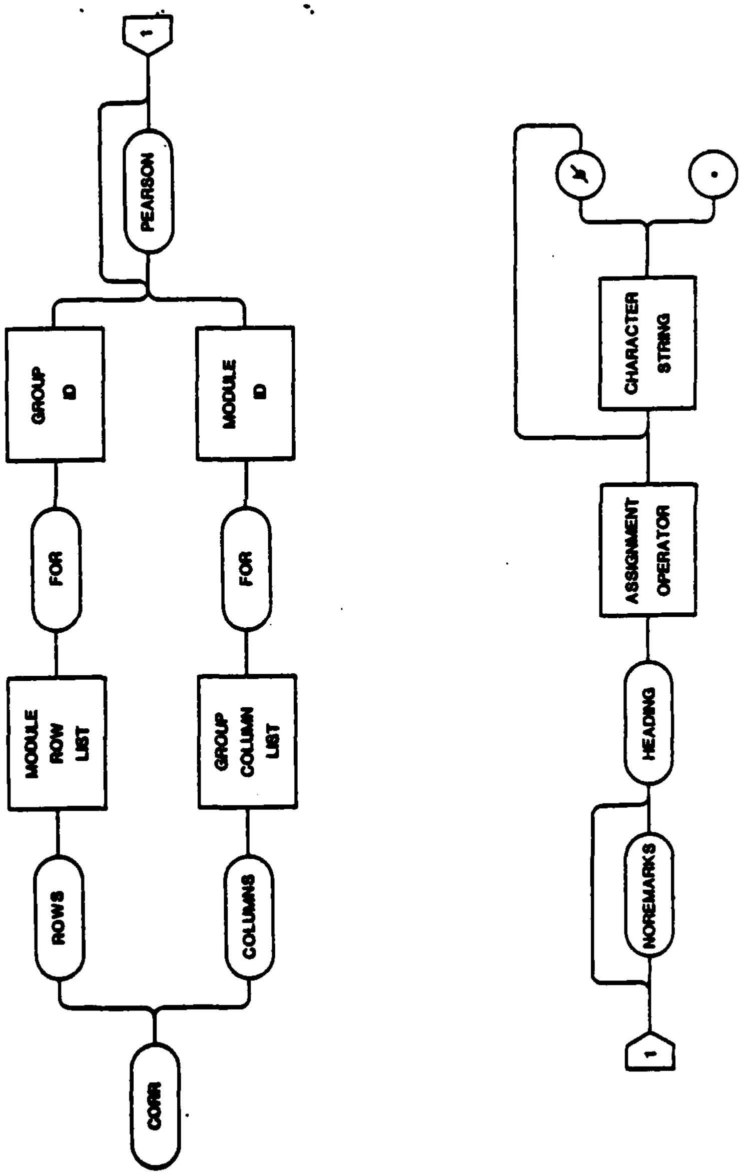
EXAMPLE 2

```
CORR COLUMNS (G4) FOR TASKS PEARSON NOREMARKS
   HEADING:='EXAMPLE 2'
              'CORRELATION MATRIX OF COLUMNS CONTAINED IN'
              'CLUSTER GROUP 4 (G4) — I1, I2, & I3'
              'CALCULATED ACROSS ALL TASKS'.
```

The above CORR statement syntax is requesting that the columns identified by the cluster group ID G4 (I1, I2, & I3) be correlated across all tasks in the study (the module ID TASKS is a CODAP80 reserved keyword). The keyword NOREMARKS has been specified.

EXAMPLE 2
PRINTED OUTPUT

```
                                                              PAGE - 1

                        STUDY ID - SAMPLEDATA80
             CORRELATION COEFFICIENTS/NUMBER OF OBSERVATIONS
                              EXAMPLE 2
               CORRELATION MATRIX OF COLUMNS CONTAINED IN
                  CLUSTER GROUP 4 (G4) -- I1, I2, & I3
                      CALCULATED ACROSS ALL TASKS

                          I - 1          I - 2          I - 3
                        ---------      ---------      ---------

          I - 1          1.00000        0.00410       -0.42675
                               5              5              5

          I - 2          0.00410        1.00000        0.67732
                               5              5              5

          I - 3         -0.42675        0.67732        1.00000
                               5              5              7
```


CORR
ROWS
COLUMNS
MODULE ROW LIST
GROUP COLUMN LIST
FOR
FOR
GROUP ID
MODULE ID
PEARSON
1
1
NOREMARKS
HEADING
ASSIGNMENT OPERATOR
CHARACTER STRING
¢
•

CREATE

INTRODUCTION

PURPOSE

The CREATE procedure is used to generate new rows or columns on the database. The new rows or columns are calculated from existing information that resides on the database.

FORM

The general form of the CREATE procedure is as follows:

1) The procedure keyword CREATE.
2) A keyword ROW or COLUMN designating what is to be created.
3) A group or module designation indicating the "length" or number of observations the created row or column will have.
4) A designation of the mathematical relationship between the new row or column being created and previously existing rows or columns.
5) An indication of whether the created data is to permanently reside on the database.
6) A remark followed by either a period or semicolon.

EXAMPLE

```
BEGIN SAMPLEDATA80 EXECUTE.
CREATE ROW FOR INCUMBENTS
   MTHS_ON_JOB = H3/12 'NUMBER OF MONTHS ON JOB'.
END.
```

The above CREATE syntax will calculate, for every incumbent on the database, the number of years they've been on the job (H3, see Sample Database) divided by 12. The new row will be named MTHS_ON_JOB and will be permanently saved on the database.

OUTPUT FORM PROCEDURE

Execution of the CREATE procedure produces no printed output. As a result of its execution, the CREATE procedure will optionally save a new row or column on the database.

CREATE SYNTAX

Refer to the syntax graph of the CREATE procedure.

CREATE

The keyword CREATE identifies the procedure.

DATA TYPE DESIGNATION

The keyword ROW or COLUMN indicates whether a row or column is to be created.

FOR

The FOR keyword alerts the CREATE procedure to expect a following group or module ID.

GROUP ID

A group ID is a defined aggregate of database columns. If the data type designation preceding the FOR keyword was ROW, then a group ID must follow. The group ID serves to indicate the "length" or number of elements the created row will have.

MODULE ID

A module ID is a defined aggregate of database rows. If the data type designation preceding the FOR keyword was COLUMN, then a module ID must follow. The module ID serves to indicate the "length" or number of elements the created column will have.

FULL ASSIGNMENT CLAUSE

A full assignment clause defines the mathematical relationship between existing data to be used in creating a new row or column. Data types may not be mixed; that is, if a row is being created then the existing data must be rows also. Relationships may be defined through the use of IF-THEN-ELSE constructs (see CREATE examples 4 & 5) or arithmetic expressions (see CREATE examples 1, 2, & 3). Acceptable arithmetic operators are addition (+), subtraction (-), division (/), multiplication (*) and exponentiation (**).

NOSAVE

Specification of the optional keyword NOSAVE indicates that the created row or column is to exist only for the duration of the current computer run.

REMARK

A remark is a string of up to 240 characters, enclosed in single quotes. The remark, which must appear, will be associated with the new ID that was generated in the full assignment clause.

PERIOD OR SEMICOLON

A period must end the syntax of the CREATE procedure. If syntax is terminated with a semicolon, specification of another series of CREATE syntax may begin without having to repeat the procedure keyword (see CREATE example 3).

CREATE EXAMPLES

EXAMPLE 1

```
BEGIN SAMPLEDATA80 EXECUTE.
CREATE ROW FOR INCUMBENTS
     H2_SQUARED = H2**2 'AGE SQUARED'.
PRINT COLUMNS (INCUMBENTS) NOREMARKS/
     ROWS (H2 H2_SQUARED)
     HEADING = 'PRINT OF AGE & AGE**2'.
END.
```

The CREATE procedure syntax in example 1 will result in a new row (named H2_SQUARED) being permanently added to the database. The new row will be 7 elements long (1 for each incumbent). The relevant calculations are displayed below:

	I1	I2	I3	I4	I5	I6	I7
H2	19	23	.	41	27	53	.
H2_SQUARED	361	529	.	1681	729	2809	.

The PRINT procedure syntax would produce a printed listing of the data values just processed by the CREATE procedure.

EXAMPLE 2

```
BEGIN SAMPLEDATA80 EXECUTE.
DESCRIBE ROWS TASKS FOR (MALES FEMALES)
     PCNTPERF = PCNT '% PERFORMING TASKS'.
CREATE COLUMN FOR TASKS
     DIFFPERF = PCNTPERF1 - PCNTPERF2
     'DIFFERENCE IN % PERFORMING TASKS - SEX'.
END.
```

The DESCRIBE procedure in the above run sequence is "describing" the tasks of the male and female incumbents in the database. It is assumed that the group IDs MALES and FEMALES were selected and saved during an earlier run stream (see SELECT example 2). Two columns (named PCNTPERF1 and PCNTPERF2) will be created from the execution of the DESCRIBE syntax. The CREATE procedure is then calculating the difference between the two columns and generating the result as a column named DIFFPERF. The column DIFFPERF will have 5 elements (1 per task). The syntax appearing in example 2 is often used as an intermediate step in the generation of a group difference description. The relevant calculations of example 2 appear below:

	PCNTPERF1	PCNTPERF2	DIFFPERF
T1	80.00	50.00	30.00
T2	100.00	50.00	50.00
T3	60.00	100.00	-40.00
T4	40.00	100.00	-60.00
T5	40.00	50.00	-10.00

EXAMPLE 3

```
     BEGIN SAMPLEDATA80 EXECUTE.
     INPUT COLUMN FOR TASKS
          TASKDIFF 'TASK DIFFICULTY'
          FORMAT '(5F1.0)'.
     CREATE COLUMN FOR TASKS
          TASKDIFF10 = TASKDIFF + 10
          'TASK DIFFICULTY PLUS 10';
        COLUMN FOR TASKS
          I1WEIGHTED = I1 * TASKDIFF10
          'INCUMBENT 1 WEIGHTED BY TASKDIFF10'.
     END.
     15296
```

The INPUT syntax in example 3 is requesting that a column be added to the database. The column will consist of task difficulty indices and will be named TASKDIFF (the values to be added appear directly after the END statement). The CREATE syntax immediately following the INPUT procedure will add 10 to every value of TASKDIFF and in so doing generate a column named TASKDIFF10. Column TASKDIFF10 will then be used in the next execution of the CREATE procedure (without having to repeat the procedure keyword due to the trailing semicolon) to weight the values in column I1, resulting in another column named I1WEIGHTED. Relevant statistics appear below:

	I1	TASKDIFF	TASKDIFF10	I1WEIGHTED
T1	64	1	11	704
T2	9	5	15	135
T3	9	2	12	108
T4	18	9	19	342
T5	0	6	16	.

EXAMPLE 4

```
     BEGIN SAMPLEDATA80 EXECUTE.
     CREATE ROW FOR G6
          IF T1=0 THEN NEWROW = 1 ELSE NEWROW = T1*2
          'ROW BASED ON T1'.
     END.
```

The CREATE syntax in example 4 illustrates the use of an IF-THEN-ELSE construct in a full assignment clause. The result of the syntax is to create a new row (named NEWROW) that will have an element for every incumbent on the database (as indicated by the system cluster group G6). If the corresponding value of T1 is zero, then NEWROW will equal 1. If the value of T1 is anything other than zero, then NEWROW will equal T1 multiplied by 2. Relevant calculations appear below:

	I1	I2	I3	I4	I5	I6	I7
T1	64	11	0	11	24	36	0
NEWROW	128	22	1	22	48	72	1

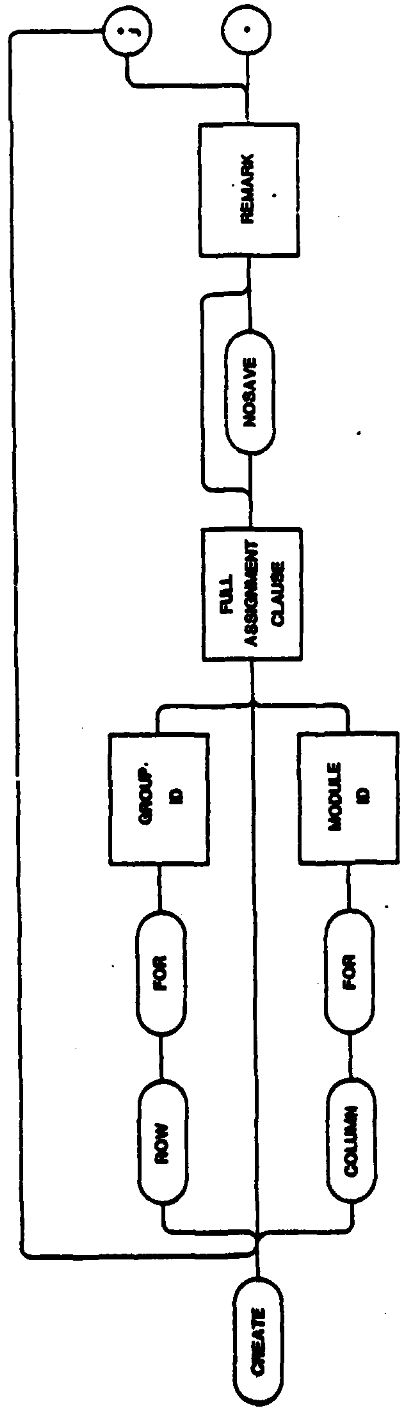
EXAMPLE 5

```
BEGIN SAMPLEDATA80 EXECUTE.
CREATE ROW FOR INCUMBENTS
     IF H3 > 30 THEN TIME_LEVEL = 4
     IF H3 > 20 THEN TIME_LEVEL = 3
     IF H3 > 10 THEN TIME_LEVEL = 2
         ELSE TIME_LEVEL = 1 NOSAVE
     '1=1-10 YEARS; 2=11-20; 3=21-30; 4=>30'.
VARSUM ROWS (TIME_LEVEL) FOR (INCUMBENTS)
     COUNT PERCENT
     HEADING = 'DISTRIBUTION OF TIME_LEVEL'.
END.
```

The CREATE syntax in example 5 is illustrating the means by which values representing intervals may be generated. The effect of the syntax is to create a new row (named TIME_LEVEL) with a value for every incumbent. The appearance of the NOSAVE keyword indicates that TIME_LEVEL will exist only for the duration of the job. The VARSUM procedure will generate a report displaying the distribution of TIME_LEVEL across all incumbents. Relevant statistics appear below:

	I1	I2	I3	I4	I5	I6	I7
H3	1	2	11	19	3	30	16
TIME_LEVEL	1	1	2	2	1	3	2

CREATE


CREATE
ROW
FOR
GROUP-ID
COLUMN
FOR
MODULE ID
FULL ASSIGNMENT CLAUSE
NOSAVE
REMARK
;
.

DESCRIBE

INTRODUCTION

PURPOSE

The DESCRIBE procedure will compute statistics on any specified aggregate of database rows or columns (a module or group) measured across one or more specified aggregates of database columns or rows (a group or module list). In particular, DESCRIBE may be used to generate statistical summarizations (such as percent performing, average, etc.) of incumbent responses to historical, task or secondary questions.

In addition to DESCRIBE's ability to calculate statistics on database rows measured across columns (the usual type of processing when generating occupational job descriptions), the procedure may also be used to "describe" database columns measured across rows. This capability gives DESCRIBE the feature of symmetry, in that any processing performed on rows across columns may also be performed on columns across rows.

FORM

The general form of the DESCRIBE command is as follows:

1) The procedure keyword DESCRIBE.
2) The keyword ROWS or COLUMNS - this keyword alerts the system that either rows or columns of the database are to be "described".
3) An indication of which rows (in the form of a module ID) or columns (in the form of a group ID) of the database are to be "described".
4) A description of at least one column or row aggregate (specified in the form of a group or module list) across which row or column statistics are to be calculated.
5) A new ID. The new ID will have a numeric value, ranging from 1 to the number of group or module IDs specified in 4, appended to it by the system. If only one column or row aggregate (a group or module ID) is specified in 4, then a numeric value is not appended to the new ID.
6) One of the statistical functions: AVGP, AVGA, STDP, STDA, PCNT, SUM or N. The function specified defines the type of statistic DESCRIBE will compute on the rows or columns specified in 3.
7) Optionally, the keyword NOSAVE.
8) Descriptive text (a remark) supplied by the user that will be be associated with the new column or row IDs added to the database.
9) A period or a semicolon. Specification of a period ends the DESCRIBE command. If, instead, a semicolon is specified, a different statistical function may be defined for the same database subset by repeating 5-9.

EXAMPLE

```
DESCRIBE ROWS TASKS FOR (INCUMBENTS)
  INCNUM := N
  'NUMBER RESPONDING TO TASKS -- ACROSS INCUMBENTS'.
```

The above DESCRIBE command syntax will calculate, for every task row (as defined by the CODAP80 system module TASKS), the number of non-zero responses across all incumbent columns in the database (as defined by the CODAP80 system group INCUMBENTS). The same effect would have been achieved had the user specified the system cluster group G6. Resulting from the execution of the above syntax, a column, five elements long (one per task) and containing the number of non-zero responses to each of the task rows across incumbents, will be permanently saved on the database. The column will be assigned the ID INCNUM as well as the descriptive remark NUMBER RESPONDING TO TASKS -- ACROSS INCUMBENTS for future reference.

OUTPUT FROM PROCEDURE

Execution of the DESCRIBE procedure produces no printed output. For every aggregate of database columns or rows (groups or modules) specified in the group or module list (which defines that part of the database across which calculations are to be performed), DESCRIBE will add a new column or row to the database. A listing of the new created row or column may be produced by appropriately referencing the ID in the syntax of the PRINT procedure (see example 3 of PRINT).

DESCRIBE SYNTAX

Refer to the syntax graph of the DESCRIBE procedure.

DESCRIBE

The keyword DESCRIBE identifies the command.

DATA TYPE DESIGNATION

The keyword ROWS or COLUMNS indicates to the system whether it is to be rows or columns of the database that are to be "described".

MODULE ID

A module ID is an identified aggregate of database rows. If the preceding data type designation was ROWS, then a module ID must follow. The module ID may be one previously defined through the use of the SELECT procedure, or may be one of the CODAP80 system modules HVARS, TVARS, TASKS or SVARS. The module ID specification serves to identify to the *DESCRIBE* procedure the database rows upon which statistics are to be calculated.

GROUP ID

A group ID is an identified aggregate of database columns. If the preceding data type designation was COLUMNS, then a group ID must follow. The group ID may be one previously defined through the use of the SELECT procedure, one of the CODAP80 system cluster groups (as defined at database creation time by the OGROUP routine) or the CODAP80 system group INCUMBENTS. The group ID specification serves to identify to the DESCRIBE procedure the database columns upon which statistics are to be calculated.

FOR

The FOR keyword alerts the DESCRIBE procedure to expect a following group or module list.

GROUP LIST

A group list is a list of at least one group ID enclosed in parentheses. Created group IDs (such as would be generated by SELECT), CODAP80 system group IDs (such as the keywords INCUMBENTS or INCS) and system cluster groups (such as G1-G3, as defined by clustering at database creation time) may all appear in a group list. Each group ID specified in the group list represents a different aggregate of database columns across which statistics for a row are to be calculated. If the preceding data type designation was ROWS, then a group list specification must follow the FOR keyword.

MODULE LIST

A module list is a list of at least one module ID enclosed in parentheses. Created module IDs (such as would be generated by SELECT) and CODAP80 system modules (such as HVARS, TVARS, TASKS and SVARS) may all appear in a module list. Each module ID specified in the module list represents a different aggregate of database rows across which statistics for a column are to be calculated. If the preceding data type designation was COLUMNS, then a module list specification must follow the FOR keyword.

ID

A user supplied "seed" ID. Any valid CODAP80 ID may be specified. The new ID will have a numeric value, ranging from 1 to the number of group or module IDs appearing in the group or module list, appended to it by the system (unless, of course, only a single aggregate ID appeared in the group or module list). The user must be careful not to specify an ID that will conflict with one previously defined in the database. The user must also take care to specify an ID that, when the numeric value is appended to it by the system, is not longer than 12 characters.

ASSIGNMENT OPERATOR

Either of the symbols '=' or ':='. Either of these symbols may be used to separate the seed ID from the statistical function that follows.

STATISTICAL FUNCTIONS

The statistical function specified defines the type of statistical operation performed by DESCRIBE on the rows or columns associated with the module or group ID designated in the syntax. The seven acceptable statistical function keywords are as follows:

AVGP	-	Average, excluding missing values.
*AVGA	-	Average, including missing values.
STDP	-	Standard deviation, excluding missing values.
*STDA	-	Standard deviation, including missing values.
*PCNT	-	Percentage of non-missing values.
SUM	-	Sum of non-missing values.
*N	-	Number of non-missing values.

*If a calculation is being performed on task rows across incumbent columns or on incumbent columns across task rows, zeros are interpreted as missing.

NOSAVE

Specification of the optional keyword NOSAVE indicates that any new rows or columns generated through the execution of the DESCRIBE procedure are not to be permanently saved for future reference.

REMARK

This is a string of up to 240 characters, enclosed in single quotes. The remark will be associated with the new row or column IDs generated. A remark must be associated with the new IDs.

PERIOD OR SEMICOLON

A period ('.') must end the syntax of the DESCRIBE procedure. If the user desires to calculate more than one type of statistic on the same database subset, the command syntax may be terminated with a semicolon, followed by the specification of a new ID, statistical function and a remark (see DESCRIBE example 1).

DESCRIBE EXAMPLES

EXAMPLE 1

```
DESCRIBE ROWS TASKS FOR (G6)
     G6AVGA := AVGA
        'AVERAGE (ALL) PER TASK--G6';
     G6AVGP := AVGP
        'AVERAGE (PERFORMING) PER TASK--G6'.
```

The above DESCRIBE command syntax will calculate, for each task row on the database (as designated by the CODAP80 system module TASKS), the average including missing values (AVGA) and the average excluding missing values (AVGP) across all the incumbent columns of the database (I1-I7, as indicated by the system cluster group G6). Execution of the above syntax will result in two columns (named G6AVGA and G6AVGP), each five elements long (one per task), being permanently added to the database. The remark AVERAGE (ALL) PER TASK--G6 will be associated with the new created column G6AVGA and the remark AVERAGE (PERFORMING) PERTASK--G6 will be associated with the second created column G6AVGP.

The values that will be calculated for the two created columns are as follows (see Sample Database):

```
G6AVGA (1) = (64+11+0+11+24+36+0)/7 = 20.86

G6AVGA (2) = (9+11+0+44+24+64+43)/7 = 27.86

G6AVGA (3) = (9+22+20+0+18+0+57)/7  = 18.00

G6AVGA (4) = (18+56+50+22+0+0+0)/7  = 20.86

G6AVGA (5) = (0+0+30+22+35+0+0)/7   = 12.43


G6AVGP (1) = (64+11+11+24+36)/5    = 29.20

G6AVGP (2) = (9+11+44+24+64+43)/6  = 32.50

G6AVGP (3) = (9+22+20+18+57)/5     = 25.20

G6AVGP (4) = (18+56+50+22)/4       = 36.50

G6AVGP (5) = (30+22+35)/3          = 29.00
```

EXAMPLE 2

```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT ROWS SHAKEDOWN (T2-T3)
  'SHAKE DOWN TASK MODULE'.
DESCRIBE ROWS SHAKEDOWN FOR (G2-G4)
  G2G4PCNT := PCNT
  'PERCENT PERFORMING--MODULE SHAKEDOWN--CLUSTERS G2-G4'.
END.
```

The above syntax specification represents a complete run stream in the CODAP80 language. The SELECT command is "selecting" two task rows (T2 and T3) to be in module SHAKEDOWN. The DESCRIBE command syntax will calculate, for each row defined to be in module SHAKEDOWN, the percentage of non-missing values (PCNT) across each of the database column aggregates contained in the CODAP80 system cluster groups G2, G3 and G4 (G2--I4, I5; G3--I4, I5, I6; G4--I1, I2, I3). Execution of the DESCRIBE syntax will result in three columns (named G2G4PCNT1, G2G4PCNT2 and G2G4PCNT3 respectively; the terminating numeral being appended to coincide with the number of group IDs specified in the group list), each two elements long (one for each row defined to be in module SHAKEDOWN), being permanently added to the database. The remark PERCENT PERFORMING--MODULE SHAKEDOWN--CLUSTERS G2-G4 will be associated with each of the three created columns. The values that will be calculated for the three created columns are as follows:

	G2G4PCNT1	G2G4PCNT2	G2G4PCNT3
T2	100.00	100.00	66.67
T3	50.00	33.33	100.00

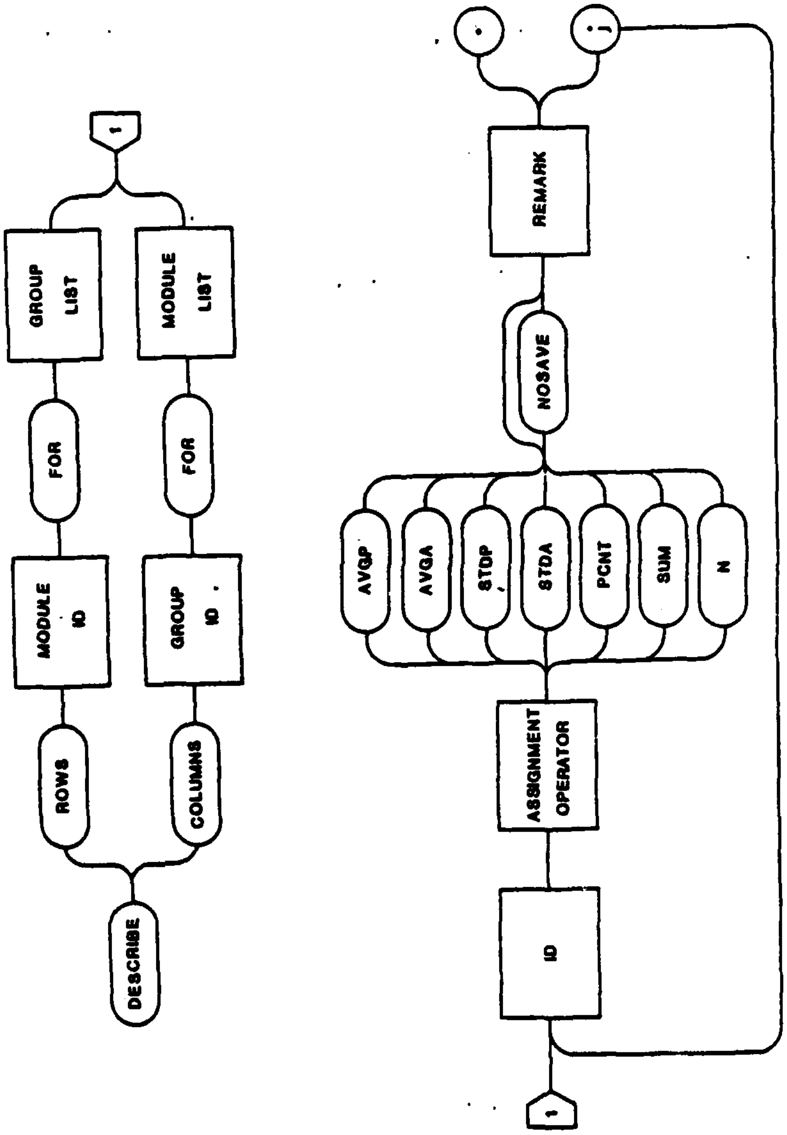
EXAMPLE 3

```
DESCRIBE COLUMNS G6 FOR (TASKS)
  G6NTASKS := N
  'NUMBER OF TASKS RESPONDED FOR EACH INCUMBENT'.
```

The above syntax illustrates the DESCRIBE procedure's symmetric capability. Examples 1 and 2 requested that the procedure "describe" rows measured across columns. Example 3 is requesting that the procedure "describe" columns measured across rows. Specifically, the above DESCRIBE command syntax will calculate, for every incumbent column on the database (as designated by the CODAP80 system cluster group G6), the number of non-missing values across all the task rows of the database (T1-T5, as indicated by the CODAP80 system module TASKS). Execution of the syntax in example 3 will result in one row being permanently added to the database (the row will be named G6NTASKS and will be seven elements long).

The values that will be calculated for the created row are as follows:

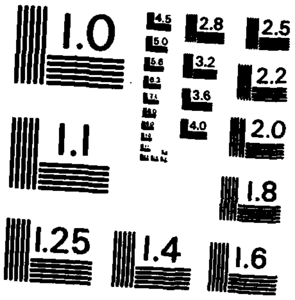
	I1	I2	I3	I4	I5	I6	I7
G6NTASKS	4	4	3	4	4	2	2


DESCRIBE
ROWS
COLUMNS
MODULE ID
GROUP ID
FOR
FOR
GROUP LIST
MODULE LIST
1
1
ID
ASSIGNMENT OPERATOR
AVGP
AVGA
STDP
STDA
PCNT
SUM
N
NOSAVE
REMARK
.
;

AD-A144 125 COMPREHENSIVE OCCUPATIONAL DATA ANALYSIS PROGRAMS 80 (CODAP80) USER'S MANUAL(U) NAVY OCCUPATIONAL DEVELOPMENT AND ANALYSIS CENTER WASHINGTON DC JAN 84 2/2

UNCLASSIFIED DOD/DF-84/006A F/G 9/2 NL

END
DATE
FILMED
9 84



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

END

INTRODUCTION

PURPOSE

The purpose of the END command is to delineate the end of a CODAP80 source language program. The END command occurs only once and is always the last statement in any CODAP80 source language program.

FORM

The general form of the END statement is as follows:

1) The procedure keyword END.
2) A period ('.').

EXAMPLE

```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT ROWS ALLSYSROWS (H1-H4, T1-T5, S1-S5)
    'ALL SYSTEM ROWS ON DATABASE'.
END.
```

The above CODAP80 source language statements represent a complete run stream that would be submitted to the computer. The example illustrates the recommended form and placement of the END command.

OUTPUT FROM PROCEDURE

Execution of the END procedure produces no printed output. The END procedure is not a procedure in the sense that, say CLUSTER or STANDARD are procedures. END performs no calculations on values in the database. END serves only to terminate a CODAP80 source language program.

END SYNTAX

Refer to the syntax graph of the END procedure.

END

The keyword END identifies the command.

PERIOD

A period ('.') must terminate the END command.

END EXAMPLES

EXAMPLE 1

```
BEGIN SAMPLEDATA80 EXECUTE.
CORR ROWS (S1-S5) for G6
   HEADING = 'CORRELATION OF S1-S5'
             'ACROSS ALL INCUMBENTS'.
END.
```

The above source statements represent a complete run stream in the CODAP80 language. Example 1 illustrates the recommended form and placement of the END command.

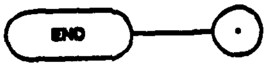
EXAMPLE 2

```
BEGIN SAMPLEDATA80 EXECUTE.
PRINT COLUMNS (G6) NOREMARKS / ROWS (H1)
   HEADING = 'EXAMPLE OF PRINT'.
END
```

Example 2 illustrates a common error in the specification of the END command. The user has neglected to terminate the END command with a period. *CODAP80 will alert the user to this fact by printing the following error* message:

```
*****************************************************************************
UNEXPECTED TERMINATION OF CODAP80 SOURCE PROGRAM FOUND.  END
COMMAND MUST BE FOLLOWED BY A PERIOD.
*****************************************************************************
```

END



INPUT

INTRODUCTION

PURPOSE

The INPUT procedure adds a new row or column to the database. The INPUT procedure is very useful for adding information to the database that was not available when the database was originally created. For example, suppose you want to classify the incumbents of a study into two categories--those who have had training and those who have not had training. By adding a new row consisting of a binary indication of training (1 if they've had training, and Ø if they have not), statistics may then be calculated across incumbents as a function of this.

FORM

The general form of the INPUT statement is as follows:

1) The procedure keyword INPUT.
2) A data type designation specifying whether a row or a column is being added to the database.
3) A designation of the aggregate of database columns the row is being added for, or a designation of the aggregate of database base rows the column is being added for.
4) A user supplied valid CODAP80 ID associated with the added row or column.
5) A user supplied FORTRAN format for reading-in the row or column values to be added to the database.
6) Options controlling the permanence of the added ID, and missing value considerations.

EXAMPLE

```
BEGIN SAMPLEDATA80 EXECUTE.
INPUT ROW FOR G6 TRAINING
    'NEW ROW NAMED TRAINING'
    FORMAT '(7F1.0)'.
END.
1110101
```

In this example, a new row named TRAINING is being added to the database. There will be a value of TRAINING for every column associated with the group ID G6 (I1-I7, see Sample Database). The string NEW ROW NAMED TRAINING, enclosed in single quotes, is the remark to be associated with the row named TRAINING. The keyword FORMAT signifies that the row ID TRAINING is to be read with the following format specification (in this case 7F1.0, indicating that the row ID TRAINING consists of 7 one digit numbers) that is enclosed in single quotes and parentheses. For an explanation of format specifications (such as 7F1.0) consult any introductory FORTRAN text.

OUTPUT FROM PROCEDURE

Execution of the INPUT procedure produces no printed output. The result of executing the INPUT procedure will be a new row or column optionally added tc the database.

INPUT SYNTAX

Refer to the syntax graph of the INPUT procedure.

INPUT

The keyword INPUT identifies the command.

DATA TYPE DESIGNATION

The keyword RCW or COLUMN indicates whether the data being added is a conceptual row or column of the database.

FOR

The FOR keyword alerts the INPUT procedure to expect a following group or module ID.

GROUP ID

A group ID is an identified aggregate of database columns. If the preceding data type designation was ROW, then a group ID must follow the FOR keyword. The group ID may be one previously defined through the use of the SELECT procedure, one of the CODAP80 system cluster groups (as defined at database creation time by the OGROUP routine) or the CODAP80 system group INCUMBENTS. The group ID specification serves to indicate to the INPUT procedure the database columns for which the new row is being added. The group ID also serves to indicate the "length" or number of elements the added row will have.

MODULE ID

A module ID is an identified aggregate of database rows. If the preceding data type designation was COLUMN, then a module ID must follow the FOR keyword. The module ID may be one previously defined through the use of the SELECT procedure, or may be one of the CODAP80 system modules HVARS, TVARS, TASKS or SVARS. The module ID specification serves to indicate to the INPUT procedure the database rows for which the new column is being added. The module ID also serves to indicate the "length" or number of elements the added column will have.

ID

This is any valid CODAP80 ID supplied by the user that will be associated with the added row or column.

NOSAVE

Specification of the optional keyword NOSAVE indicates that the added row or column will exist on the database only for the duration of the computer run.

REMARK

This is a string of up to 240 characters enclosed in single quotes. The remark will be associated with the added row or column. A remark must be associated with the added row or column.

MISSING
ASSIGNMENT OPERATOR
CONSTANT

Some of the elements of the row or column to be added to the database may be missing (as opposed to being zero or blank). To signal the INPUT procedure that a given value is missing, choose a unique integer constant as the identifier in the missing option. For example, suppose the user was adding a new row to the database, and one of its five elements was missing. By indicating a unique integer constant in the missing option (let's say 99), the INPUT procedure would then know that any values of 99 that were input as the new row should be set to missing (see INPUT example 1).

FORMAT

The FORMAT keyword serves to indicate to the INPUT procedure that the following string enclosed in single quotes is to be used as the input format for reading-in the values of the row or column to be added.

FORMAT SPECIFICATION

The format specification for the INPUT procedure may be any valid 1966 Ansi Standard FORTRAN format in parentheses, enclosed in single quotes. The format will be used by the INPUT procedure to read-in the values of the added row or column. The place in the input stream of a CODAP80 source language program where the values of the row or column to be added are to appear is directly after the terminating END statement (see INPUT examples 1 and 2). For an explanation of FORTRAN formats, consult any introductory FORTRAN text.

PERIOD

A period ('.') must end the INPUT statement.

INPUT EXAMPLES

EXAMPLE 1

```
BEGIN SAMPLEDATA80 EXECUTE.
INPUT ROW FOR G6 RACE
  'RACIAL BACKGROUND OF INCUMBENTS'
  MISSING := 9 FORMAT '(7F1.0)'.
END.
1192319
```

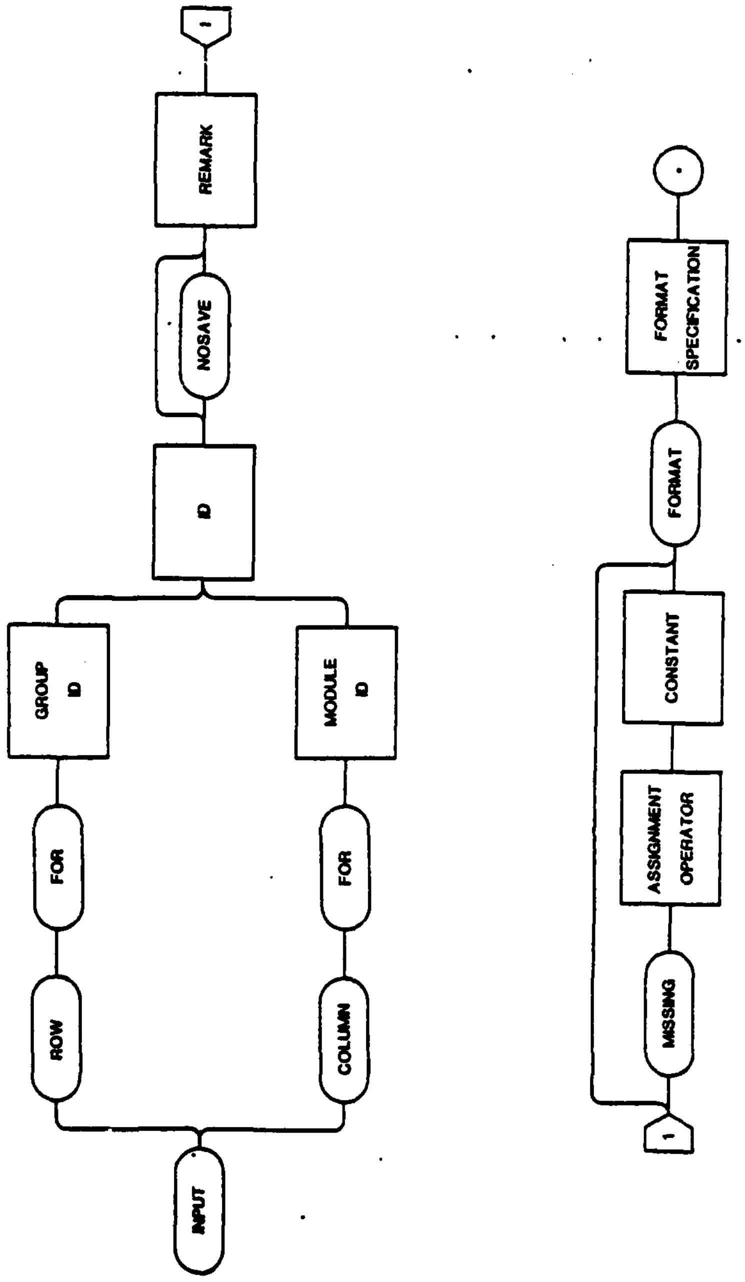
The above syntax represents a complete run stream in the CODAP80 language. The INPUT syntax is requesting that a new row (to be named RACE) be added to the database. The row will have an element for every incumbent column on the database (as defined by the CODAP80 system cluster group G6) and will be associated with the remark RACIAL BACKGROUND OF INCUMBENTS. Two of the seven race values are missing and the syntax is alerting the INPUT procedure to set to missing any values of the row to be added that equal 9. The format specification indicates that the added row consists of seven 1-digit numbers.

EXAMPLE 2

```
BEGIN SAMPLEDATA80 EXECUTE.
INPUT COLUMN FOR TASKS RANKING
  'ALTERNATE RANKING FOR TASKS'
  FORMAT '(5F1.0)'.
END.
21453
```

The above syntax represents a complete run stream in the CODAP80 language. The INPUT syntax is requesting that a new column (to be named RANKING) be added to the database. The new column will have an element for every task row on the database (as defined by the CODAP80 system module TASKS) and will be associated with the remark ALTERNATE RANKING FOR TASKS. The format specification indicates that the added column consists of five 1-digit numbers.

The rationale for the operation shown in example 2 might be that the user wishes to see the tasks on a database sorted on some arbitrary dimension. After the column of rankings was added, the user could then have the print procedure display the tasks, sorted by the newly added column.


INPUT
ROW
FOR
GROUP ID
COLUMN
FOR
MODULE ID
ID
NOSAVE
REMARK
1
1
MISSING
ASSIGNMENT OPERATOR
CONSTANT
FORMAT
FORMAT SPECIFICATION
.

PRINT

INTRODUCTION

PURPOSE

The PRINT procedure displays information that exists in the database. In addition, various summary statistics are optionally calculated and displayed.

FORM

The general form of the PRINT statement is as follows:

1) The procedure keyword PRINT.
2) A description of which part of the database is being use to define the vertical axis.
3) A description of which part of the database is being used to define the horizontal axis.
4) A description of what is to be printed as a title at the top of the produced report.
5) Various options that define operations to be performed on the displayed information and that control the appearance of the output.

EXAMPLE

```
PRINT ROWS (TASKS) AVGP / COLUMNS (I1-I3)
     HEADING := 'EXAMPLE OF PRINT PROCEDURE'.
```

This PRINT statement would output tasks as designated by the CODAP80 system module TASKS down the vertical axis (any designation occurring before the slash (/) indicates the elements of the vertical axis). The keyword AVGP specifies that the average (non-missing elements only) is to be calculated on the elements occurring down the vertical axis. I1-I3 specifies that the first three columns of the database are to comprise the elements of the horizontal axis (any designation occurring after the (/) indicates the elements of the horizontal axis). The string enclosed in single quotes following the HEADING keyword indicates what is to be printed at the top of the page as a title.

OUTPUT FROM PROCEDURE

Execution of the PRINT procedure produces a report displaying the rows and columns of a database. Exactly which rows and columns are displayed, and the appearance the output will have, is a function of user input.

CAUTION: The user is warned to display great care when requesting output from the PRINT procedure. Inadvertent requests could conceivably generate a report consisting of inordinate amounts of paper. For example, in a study with 1,000 incumbents, measured on 200 tasks, that had been clustered, the following PRINT statement would generate over 350 pages of output:

```
PRINT ROWS (TASKS) NOREMARKS /
  COLUMNS (G999) NOREMARKS MISSING
  HEADING:= 'VERY LARGE PRINTED OUTPUT'.
```

The word TASKS specifies that all tasks in the study will constitute the vertical axis and G999, as the last stage in the clustering process, indicates that all incumbents in the study will comprise the elements of the horizontal axis.

The user is also warned that the above PRINT command represents the most inefficient way to print database (system) information. A much faster PRINT command to dump the same database information is as follows:

```
PRINT COLUMNS (G999) NOREMARKS /
  ROWS (TASKS) NOREMARKS MISSING
  HEADING:= 'VERY LARGE PRINTED OUTPUT'.
```

PRINT SYNTAX

Refer to the syntax graph of the PRINT procedure.

PRINT

The keyword PRINT identifies the command.

VERTICAL DATA TYPE DESIGNATION

The PRINT procedure displays the rows and columns of a two-dimensional occupational database. Specification of the keyword ROWS as the vertical data type designation indicates that the vertical axis of the printed output is to be made up of database rows. Conversely, if the keyword specified is COLUMNS then the vertical axis of the output will consist of database columns. If the data type designation for the vertical axis is ROWS, the horizontal data type designation must be COLUMNS. The reverse would be true were columns of the database chosen to define the vertical axis.

GROUP LIST

A group list is a list of at least one group ID enclosed in parentheses. The group list serves to indicate to the PRINT procedure which database columns are to comprise the vertical axis of the printed output. If the vertical data type designation was COLUMNS, then a group list specification must follow. Group IDs appearing in the group list may consist of created groups defined through the use of the SELECT procedure, CODAP80 system cluster groups (such as G1-G3, as defined by the OGROUP routine at database creation time) and the system group INCUMBENTS.

MODULE LIST

A module list is a list of at least one module ID enclosed in parentheses. The module list serves to indicate to the PRINT procedure which database rows are to comprise the vertical axis of the printed output. If the vertical data type designation was ROWS, then a module list specification must follow. Module IDs appearing in the module list may consist of created modules defined through the use of the SELECT procedure and the CODAP80 system modules HVARS, TVARS, TASKS and SVARS.

NOREMARKS

Use of the keyword NOREMARKS at this point in the procedure's syntax indicates that any remarks associated with the IDs specified to comprise the vertical axis of the output are not to be printed.

NOSUMMARY

Specification of the keyword NOSUMMARY indicates that a separate summarization of vertical axis aggregates (group or module IDs appearing in the vertical group or module list specification) is not to be printed.

SUMONLY

The keyword SUMONLY indicates that only the summary calculations down vertical elements are to be printed. The actual vertical elements that went into the summary calculations will not be printed.

SUMMARY FUNCTIONS

There are six different summary calculations that can be performed down the vertical axis. They are as follows:

AVGA	-	Average, including missing values.
AVGP	-	Average, excluding missing values.
STDA	-	Standard deviation, including missing values.
STDP	-	Standard deviation, excluding missing values.
SUM	-	Sum of non-missing values.
N	-	Number of non-missing values.

Specification of these summary functions may occur in any order. A summary function may not be specified more than once.

SLASH

The slash delimiter '/' serves to differentiate vertical axis designations and horizontal axis designations. Designations occurring before the slash ('/') define the elements of the vertical axis and designations following the slash define the elements of the horizontal axis. If the user has specified COLUMNS before the slash, ROWS must be specified following the slash. Conversely, if the user specifies ROWS before the slash, COLUMNS must follow the slash.

HORIZONTAL DATA TYPE DESIGNATION

The keyword ROWS indicates that row elements of the database will comprise the horizontal axis of the printed output, while the keyword COLUMNS indicates that the horizontal axis will consist of column elements.

MODULE ROW LIST

A Module Row List (MROWLT) is a list of at least one module or row ID enclosed in parentheses. Lists of module IDs, system row lists and lists of row IDs may all occur together in a MROWLT. In regard to the PRINT procedure, the MROWLT serves to specify which row elements are to comprise the horizontal axis on output.

GROUP COLUMN LIST

A Group Column List (GCOLST) is a list of at least one group or column ID enclosed in parentheses. Lists of group IDs, system group lists, system column lists and lists of column IDs may all occur together in a GCOLST. In regard to the PRINT procedure, the GCOLST serves to specify which column elements are to comprise the horizontal axis on output.

NOREMARKS

Specifying NOREMARKS indicates that the remarks associated with the horizontal axis element designations are not to be printed.

MISSING

The default condition for the PRINT procedure is not to print out those elements of data that are missing (many of the task responses from an individual may be missing). To print out the missing values, the user needs to appropriately specify the keyword MISSING.

SORT DESCENDING BY

The user has the option of sorting the printed information. The default is to sort by ascending value. By specifying DESCENDING the sort proceeds by descending value.

SORT ID

The SORT ID is a single row or column ID enclosed in parentheses. The ID must agree in type with the horizontal axis element designations. In other words, if the horizontal axis data type is ROWS, then the element ID specified as the SORT ID must be a row also. If the horizontal axis data type is COLUMNS, then the element ID specified as the SORT ID must be a column also. The values of the element ID specified in the SORT ID will be used to sort the values of the horizontal axis element designations.

CUM

This keyword indicates that a running accumulation of specified IDs is to be calculated and printed.

CUM LIST

This is a list of at least one row or column ID enclosed in parentheses. For those IDs indicated in the CUMLIST, a running accumulation will be calculated and printed. The IDs appearing in the CUMLIST must agree in type with the data type of the horizontal axis designation.

NORESET

Specification of the NORESET keyword indicates that any accumulation vectors are not to be reset to zero (which is the default) at the beginning of a new vertical data type designation.

COUNT

Specification of the keyword COUNT alerts the system to expect a following integer constant, the value of which determines the occurrence of line breaks in the procedure output. Line break specification is optional.

CONSTANT

The value of the integer constant following the COUNT keyword determines the occurrence of procedure output line breaks. If, for example, the integer constant specified was the number 3, a line break and count would occur following every third line output by the procedure (see example 6, of PRINT).

NOSKIP

Specification of the NOSKIP keyword indicates that printed output is not to go to the top of a new page when printing the start of another vertical data type designation (a new group or module).

FORMAT
FORMAT SPECIFICATION

The PRINT procedure allows the user to specify the number of decimal places that are to appear with printed values. The number of decimal places that may be specified range from 0 (print as an integer) to 9. The user is warned to use good judgement when selecting a format specification. Up to 12 digits (including the sign) may be printed. Consider the following example:

```
PRINT COLUMNS (INCUMBENTS) NOREMARKS / ROWS (H1-H9)
     FORMAT H1 0 H3 1
     HEADING='PRINT WITH FORMAT'.
```

The above PRINT command would produce a PRTVAR report. History variable H1 would print as an integer and H3 would print with one decimal place. H2, H4-H9 would default to two decimal places.

HEADING

The word HEADING serves to indicate that the following string is to be used as a report title.

ASSIGNMENT OPERATOR

Either the symbols '=' or ':='. Either of these symbols may be used to separate the HEADING keyword from the title character string.

CHARACTER STRING

Up to 10 lines of 131 characters each may comprise the character string in the PRINT statement. Each title line of up to 131 characters is enclosed in single quotes, with the beginning of a new title line indicated by a blank and another line enclosed in single quotes.

For example:

```
HEADING:= 'EXAMPLE OF A HEADING STRING BEING USED TO'
'DEMONSTRATE HOW TITLES ARE CONSTRUCTED'.
```

This example would produce two title lines centered at the top of the output page:

```
EXAMPLE OF A HEADING STRING BEING USED TO
 DEMONSTRATE HOW TITLES ARE CONSTRUCTED
```

PERIOD

A period ('.') must end the PRINT statement.

PRINT EXAMPLES

EXAMPLE 1

```
PRINT ROWS (TASKS) / COLUMNS (AVGPAGE STDPAGE)
   HEADING := 'PRINT EXAMPLE 1'
     'PRINTING-OUT CREATED COLUMNS FROM'
     'AVALUE EXAMPLE 1'.
```

The reader is referred to example 1 of the AVALUE procedure. The above PRINT syntax is requesting that task rows comprise the vertical axis of the output, and the columns AVGPAGE and STDPAGE (generated through the execution of the syntax in AVALUE example 1) comprise the horizontal axis.

EXAMPLE 1
PRINTED OUTPUT

```
                                                                  PAGE - 1
                         STUDY ID - SAMPLEDATA80
                            PRINT EXAMPLE 1
                   PRINTING-OUT CREATED COLUMNS FROM
                            AVALUE EXAMPLE 1

AVGPAGE  AVERAGE AGE (AVGP), G6
STDPAGE  STD AGE (STDP), G6

                                                 AVGPAGE      STDPAGE
                                                 ---------    ---------

TASKS
------

T - 1  SUBDUE VIOLENT INMATES                      32.60        14.10
T - 2  SHAKE DOWN INMATES                          32.60        14.20
T - 3  SHAKE DOWN VISITORS                         23.00         4.00
T - 4  ESCORT INMATES                              27.70        11.70
T - 5  TESTIFY IN COURT                            34.00         9.90
```

EXAMPLE 2

```
   BEGIN SAMPLEDATA80 EXECUTE.
   SELECT ROWS MOD1 (T1-T3) 'SHAKE DOWN TASKS';
        . ROWS MOD2 (T4-T5) 'OTHER TASKS'.
   PRINT ROWS (MOD1 MOD2) AVGA STDA / COLUMNS (I1 I6) NOREMARKS
        MISSING SORT BY (I1)
        HEADING:= 'EXAMPLE 2'
            'PRINTING OFF MODULE 1 & 2 DATA FOR'
            'INCUMBENTS 1 & 6'.
   END.
```

In this example, the user is requesting that the vertical axis of the output consist of rows (specifically, the rows identified by the module MOD1 - tasks 1-3 and MOD2 - tasks 4-5), and that the summary statistics AVGA and STDA be calculated down them. The horizontal axis of the output will consist of the columns I1 and I6, all missing values will be printed and the values of these two columns will be in I1 ascending sort order. A separate module summary will be printed on the page after the actual procedure output.

EXAMPLE 2
PRINTED OUTPUT

```
                                                             Page - 1
                         STUDY ID - SAMPLEDATA80
                               EXAMPLE 2
                   PRINTING OFF MODULE 1 & 2 DATA FOR
                          INCUMBENTS 1 & 6

                                             I - 1          I - 6
                                            -------        -------
MOD1   SHAKE DOWN TASKS
----
T - 2  SHAKE DOWN INMATES                     9.00          64.00
T - 3  SHAKE DOWN VISITORS                    9.00           0.00
T - 1  SUBDUE VIOLENT INMATES                64.00          36.00
----                                        -------        -------
AVGA                                         27.33          33.00
STDA                                         31.75          32.08
```

EXAMPLE 2
PRINTED OUTPUT (continued)

```
******************************************************************************

                                                              Page - 2
                          STUDY ID - SAMPLEDATA80
                                EXAMPLE 2
                   PRINTING OFF MODULE 1 & 2 DATA FOR
                           INCUMBENTS 1 & 6

                                               I - 1          I - 6
                                              -------        -------
MOD2    OTHER TASKS
-----
T - 5   TESTIFY IN COURT                        0.00           0.00
T - 4   ESCORT INMATES                         18.00           0.00
-----                                         -------        -------
AVGA                                            9.00           0.00
STDA                                           12.73           0.00
```

EXAMPLE 2
MODULE SUMMARY
PRINTED OUTPUT

```
                                                                  PAGE - 1
                        STUDY ID - SAMPLEDATA80
                               EXAMPLE 2
                    PRINTING OFF MODULE 1 & 2 DATA FOR
                           INCUMBENTS 1 & 6

                        *** AVGA SUMMARY ***

                                       I - 1         I - 6
                                      -------       -------

MOD2    OTHER TASKS                     9.00          0.00
MOD1    SHAKE DOWN TASKS               27.33         33.33

**************************************************************************
                                                                  PAGE - 2
                        STUDY ID - SAMPLEDATA80
                               EXAMPLE 2
                    PRINTING OFF MODULE 1 & 2 DATA FOR
                           INCUMBENTS 1 & 6

                        *** STDA SUMMARY ***

                                       I - 1         I - 6
                                      -------       -------

MOD2    OTHER TASKS                    12.73          0.00
MOD1    SHAKE DOWN TASKS               31.75         32.08
```

EXAMPLE 3

Assume that the user had, in an earlier job run, created a new column on the data base with the following DESCRIBE statement (see the section on the DESCRIBE procedure for more information):

```
DESCRIBE ROWS TASKS FOR (G6)
   G6TASKSAVGP:=AVGP
   'AVERAGE TIME SPENT PERFORMING--G6'.
```

The effect of this DESCRIBE statement is to create a new column with a length NTASK (5) elements long. The new column vector is named G6TASKSAVGP and consists of the average (for those performing) across all incumbents (G6 is the last stage in the clustering process - there were a total of 7 incumbents) for each task in the study.

The values in this column (1 element for each task) would be:

```
29.2     32.5     25.2     36.5     29.0
```

Printing off this column, with all remarks, the user would code the following PRINT statement:

```
PRINT ROWS (MOD1 MOD2) NOSUMMARY SUM / COLUMNS
  (G6TASKSAVGP) HEADING:='EXAMPLE 3'
  'PRINTING OFF THE GENERATED COLUMN G6TASKSAVGP'.
```

EXAMPLE 3
PRINTED OUTPUT

```
                                                           PAGE - 1
                          STUDY ID - SAMPLEDATA80
                                EXAMPLE 3
                 PRINTING OFF THE CREATED COLUMN G6TASKSAVGP

G6TASKSAVGP    AVERAGE TIME SPENT PERFORMING--G6

                                                  G6TASKSAVGP
                                                  -----------

MOD1   SHAKE DOWN TASKS
----
T - 1  SUBDUE VIOLENT INMATES                           29.20
T - 2  SHAKE DOWN INMATES                               32.50
T - 3  SHAKE DOWN VISITORS                              25.20
----                                              -----------
SUM                                                     86.90
```

EXAMPLE 3
PRINTED OUTPUT (continued)

```
********************************************************************************

                                                                     PAGE - 2
                       STUDY ID - SAMPLEDATA80
                              EXAMPLE 3
             PRINTING OFF THE CREATED COLUMN G6TASKSAVGP

G6TASKSAVGP    AVERAGE TIME SPENT PERFORMING--G6

                                                       G6TASKSAVGP
                                                       -----------

MOD2   OTHER TASKS
-----
T - 4  ESCORT INMATES                                        36.50
T - 5  TESTIFY IN COURT                                      29.00
-----                                                  -----------
SUM                                                          65.50
```

EXAMPLE 4

For a much simpler report of example 3, the following options can be requested in the PRINT statement (an accumulation vector has also been requested):

```
PRINT ROWS (MOD1 MOD2) NOREMARKS/COLUMNS (G6TASKSAVGP)
   NOREMARKS CUM(G6TASKSAVGP)
   HEADING:='EXAMPLE 4'
   'A MORE PARSIMONIOUS REQUEST OF EXAMPLE 3'
   'ACCUMULATION VECTOR ADDED'.
```

EXAMPLE 4
PRINTED OUTPUT

```
                                                          PAGE - 1
                         STUDY ID - SAMPLEDATA80
                                EXAMPLE 4
                 A MORE PARSIMONIOUS REQUEST OF EXAMPLE 3
                        ACCUMULATION VECTOR ADDED

                                          ACCUMULATE
                   G6TASKSAVGP            G6TASKSAVGP
                   -------------          -------------
MOD1
-----

T - 1                    29.20                  29.20
T - 2                    32.50                  61.70
T - 3                    25.20                  86.90

******************************************************************

                                                          PAGE - 2
                         STUDY ID - SAMPLEDATA80
                                EXAMPLE 4
                 A MORE PARSIMONIOUS REQUEST OF EXAMPLE 3
                        ACCUMULATION VECTOR ADDED

                                          ACCUMULATE
                   G6TASKSAVGP            G6TASKSAVGP
                   -------------          -------------
MOD2
-----

T - 4                    36.50                  36.50
T - 5                    29.00                  65.50
```

EXAMPLE 5

In the previous four examples the PRINT procedure was always requested to produce a report in which the rows of the database comprised the vertical axis of the output and the horizontal axis of the output was comprised of database columns. In this example, symmetric display of the database will be addressed by instructing the PRINT procedure to output database columns on the vertical axis and data base rows on the horizontal axis.

```
PRINT COLUMNS (G4) NOREMARKS/ROWS (MOD1) NOREMARKS
HEADING:='EXAMPLE 5'  'EXAMPLE OF SYMMETRIC DISPLAY'.
```

EXAMPLE 5
PRINTED OUTPUT

```
                                                                PAGE - 1
                              STUDY ID - SAMPLEDATA80
                                    EXAMPLE 5
                            EXAMPLE OF SYMMETRIC DISPLAY

G - 4           T - 1          T - 2          T - 3
-----         --------       --------       ---------

I - 1           64.00           9.00           9.00
I - 2           11.00          11.00          22.00
I - 3            0.00           0.00          20.00
```

EXAMPLE 6

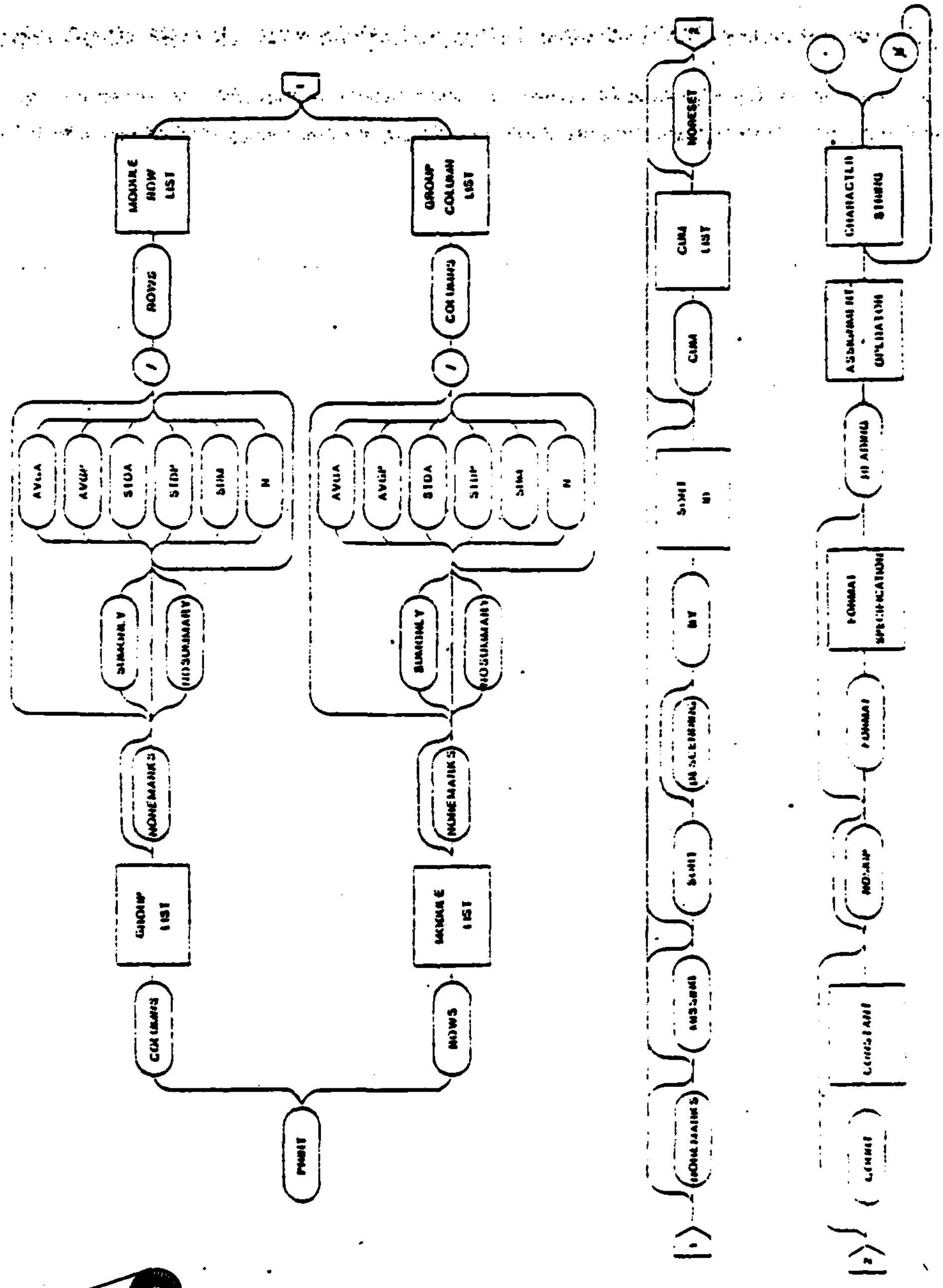
```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT ROWS ALLROWS (H1-H4, T1-T5, S1-S5)
   'MODULE CONTAINING ALL SYSTEM ROWS'.
DESCRIBE ROWS (ALLROWS) FOR INCUMBENTS
   ROWN     = N 'NUMBER RESPONDING TO ROW';
   ROWPCNT  = PCNT 'PERCENT RESPONDING TO ROW'.
PRINT ROWS (ALLROWS) NOSUMMARY AVGA /
   COLUMNS (ROWN ROWPCNT) COUNT 5
   HEADING  = 'EXAMPLE 6' 'USE OF COUNT OPTION IN PRINT'.
END.
```

The command syntax in example 6 will result in a report displaying both the number and percentage of incumbents responding to each of the system rows on the database. The SELECT command is requesting that all system rows be associated with the module ID ALLROWS. The DESCRIBE command immediately following will calculate the number and percentage of incumbent responses to each of the rows associated with the module ID ALLROWS. The two created columns (ROWN and ROWPCNT) generated from execution of the DESCRIBE syntax will each contain 14 elements and will be permanently saved on the database. The PRINT command syntax will display down the vertical axis the rows identified by the module ID ALLROWS and, across the horizontal axis of the printed output, the created columns ROWN and ROWPCNT. Note the effect of using the COUNT option.

EXAMPLE 6
PRINTED OUTPUT

```
                                                                Page - 1

                           STUDY ID - SAMPLEDATA80
                                 EXAMPLE 6
                         USE OF COUNT OPTION IN PRINT

ROWN         NUMBER RESPONDING TO ROW
ROWPCNT      PERCENT RESPONDING TO ROW

ALLROWS                                              ROWN         ROWPCNT
-----------                                       ---------    -----------
H - 1       SEX                                      7.00          100.00
H - 2       AGE                                      5.00           71.43
H - 3       YEARS ON JOB                             7.00          100.00
H - 4       INCUMBENT ID                             7.00          100.00
T - 1       SUBDUE VIOLENT INMATES                   5.00           71.43

                                                                 _______5
--------

T - 2       SHAKE DOWN INMATES                       6.00           85.71
T - 3       SHAKE DOWN VISITORS                      5.00           71.43
T - 4       ESCORT INMATES                           4.00           57.14
T - 5       TESTIFY IN COURT                         3.00           42.86
S - 1       SECONDARY - SUBDUE VIOLENT INMATES       2.00           28.57

                                                                 ______10
--------

S - 2       SECONDARY - SHAKE DOWN INMATES           6.00           85.71
S - 3       SECONDARY - SHAKE DOWN VISITORS          5.00           71.43
S - 4       SECONDARY - ESCORT INMATES               4.00           57.14
S - 5       SECONDARY - TESTIFY IN COURT             3.00           42.86

-----------                                       ---------    -----------
AVGA                                                 4.93           70.41
```

PRINT



Reproduced from best available copy.

RANDOM

INTRODUCTION

PURPOSE

From the elements of any specified module or group ID, the RANDOM procedure will randomly select a subsetting module or group.

FORM

The general form of the RANDOM command is as follows:

1) The procedure keyword RANDOM.
2) The keyword ROWS or COLUMNS.
3) An indication of the row or column aggregate (in the form of a module or group ID) from which random selection is to be made.
4) A constant or, optionally, the keyword KTH followed by a constant.
5) A new ID. The new ID will be assigned to the module or group subset selected.
6) Optionally, the keyword NOSAVE.
7) Descriptive text (a remark) supplied by the user that will be associated with the new ID.

EXAMPLE

```
RANDOM ROWS SVARS 3 RANDOMSVARS '3 RANDOM SVARS'.
```

The above RANDOM command syntax is requesting that three rows be randomly selected from the row elements of CODAP80 system module SVARS (S1-S5). The randomly selected module subset will be assigned the ID RANDOMSVARS as well as the remark 3 RANDOM SVARS.

OUTPUT FROM PROCEDURE

Execution of the RANDOM procedure produces no printed output. If NOSAVE was not specified the randomly selected module or group will be permanently saved on the database for future reference.

RANDOM SYNTAX

Refer to the syntax graph of the RANDOM procedure.

RANDOM

The keyword RANDOM identifies the command.

DATA TYPE DESIGNATION

The keyword ROWS or COLUMNS indicates to the system whether it is to be rows or columns of the database that are randomly selected.

MODULE ID

A module ID is an identified aggregate of database rows. The aggregate of rows identified by the module ID will serve as the population from which the RANDOM procedure will select a module subset. If the data type designation following the RANDOM command keyword is ROWS, then a module ID must follow.

CAUTION: All created module IDs appearing in the RANDOM command syntax must have been selected and permanently saved during a previous execution of the CODAP80 interpreter. RANDOM cannot process created modules that were selected in the same run stream.

GROUP ID

A group ID is an identified aggregate of database columns. The aggregate of columns identified by the group ID will serve as the population from which the RANDOM procedure will select a group subset. If the data type designation following the RANDOM command keyword is COLUMNS, then a group ID must follow.

CAUTION: All created group IDs appearing in the RANDOM command syntax must have been selected and permanently saved during a previous execution of the CODAP80 interpreter. RANDOM cannot process groups that were selected in the same run stream.

KTH

Appearance of the optional keyword KTH indicates that the selected module or group subset is to consist of every "Kth" element of the module or group serving as the population, with the first element being randomly chosen. See example 2 of RANDOM for more information.

CONSTANT

A user supplied integer numeric value, such as '10'. The value of the constant will determine the number of elements selected frcm the population module or group to be in the subset. *If the optional* keyword KTH preceeds the constant, then the value of the constant represents every "Kth" element to be selected.

ID

Any valid CODAP80 ID, supplied by the user. The ID supplied will be associated with the module or group subset that was randomly selected.

NOSAVE

If the optional keyword NOSAVE is specified, the randomly selected module or group will not be permanently saved for future reference.

REMARK

This is a string of up to 240 characters, enclosed in single quotes. The remark will be associated with the new module or group ID created. A remark must be associated with the new ID.

PERIOD

A period ('.') must end the RANDOM statement.

RANDOM EXAMPLES

EXAMPLE 1

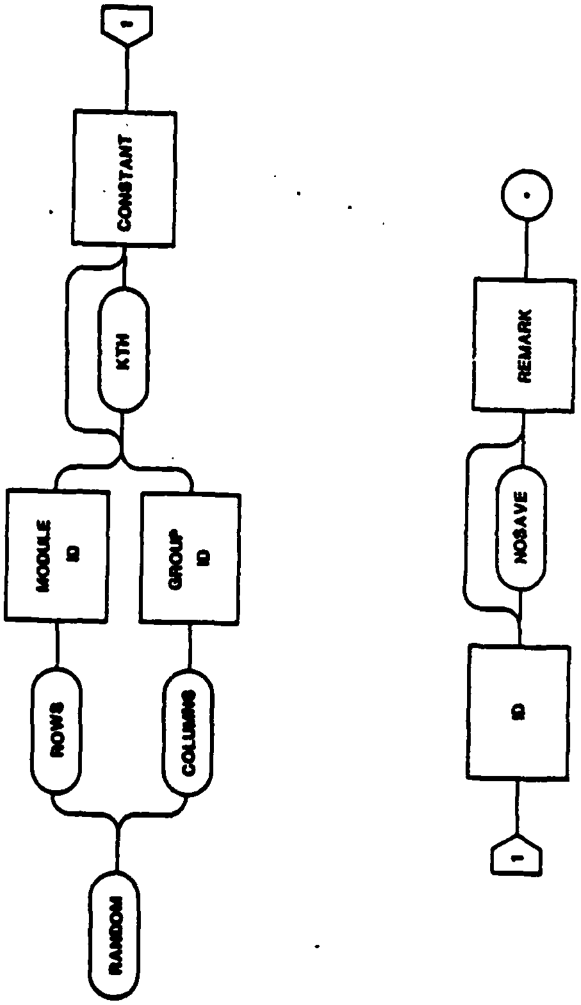
```
BEGIN SAMPLEDATA80 EXECUTE.
RANDOM ROWS TASKS'2 RANDMODULE
  '2 TASK ROWS SELECTED AT RANDOM'.
END.
```

The above RANDOM command syntax is requesting that two rows be randomly selected from the aggregate of rows identified by the module ID TASKS (T1-T5). Execution of the above syntax will result in the creation of module RANDMODULE. This module ID will be associated with two task rows selected randomly. The module ID, and its associated remark, will be permanently stored for future reference.

EXAMPLE 2

```
BEGIN SAMPLEDATA80 EXECUTE.
RANDOM COLUMNS INCUMBENTS KTH 2 RANDOMGROUP
  'EVERY 2ND INCUMBENT COLUMN'.
END.
```

The above RANDOM command syntax is requesting that from the column aggregate identified by the system group INCUMBENTS (I1-I7), every 2nd (Kth) column be selected and be identified by the new created group ID RANDOMGROUP. The first column selected from the system group INCUMBENTS is to be randomly determined. If, for example, the first column chosen randomly from group INCUMBENTS was I4, then the new created group RANDOMGROUP will be associated with 4 system columns; I4, I6, I1 and I3. The procedure will continue selecting every 2nd element of the group ID specified to act as the population until it has cycled through all the elements associated with the ID INCUMBENTS, stopping the selection process only when it happens back across the first column element (the randomly selected column, I4) that started the process to begin with.


RANDOM
ROWS
COLUMNS
MODULE ID
GROUP ID
KTH
CONSTANT
1
1
ID
NOSAVE
REMARK
.

RELY

INTRODUCTION

PURPOSE

The RELY procedure calculates reliability estimates of the mean of a set of k raters (Rkk) and that of a single rater (Rll). The reliability estimates calculated are useful in the determination of agreement among the responses from a series of raters or judges. The computational method is from Winer (1971).

FORM

The general form of the RELY procedure is as follows:

1) The procedure keyword RELY.
2) The keyword ROWS or COLUMNS.
3) A designation of the rows (a module) or columns (a group) for which reliabilities are to be calculated.
4) An indication of whether or not the reliabilities are to be "adjusted."
5) Heading(s) to serve as titles on the printed output from the procedure.

EXAMPLE

```
BEGIN SAMPLEDATA80 EXECUTE.
ADDATA COLUMNS FOR TASKS N=3
    JUDGE1  'JUDGE NUMBER 1'
    JUDGE2  'JUDGE NUMBER 2'
    JUDGE3  'JUDGE NUMBER 3'
    FORMAT  '(5F1.0)'.
SELECT COLUMNS JUDGES (JUDGE1 JUDGE2 JUDGE3)
    'GROUP OF JUDGES'.
RELY COLUMNS JUDGES FOR TASKS
    HEADING='EXAMPLE OF RELY PROCEDURE'.
END.
27441
38331
26342
```

The above example illustrates a classic use of the RELY procedure. Three judges are rating each of the five tasks on the database as to the consequences of their inadequate performance. The ratings are appended to the database through the use of the ADDATA procedure. The three judges' responses are formed into a group by the SELECT procedure and then this group of responses is submitted to the RELY procedure in order that an estimate of reliability may be calculated.

OUTPUT FROM PROCEDURE

Output from the RELY procedure consists of printed output displaying Rkk and Rll reliabilities, the various sums of squares and mean squares that went into the calculation of the reliabilities and a break-down of the individual raters' correlations and T values.

RELY SYNTAX

Refer to the syntax graph of the RELY procedure.

RELY

The keyword RELY identifies the command.

DATA TYPE DESIGNATION

The keyword ROWS indicates that the RELY procedure is to preform its calculations on database rows. The keyword COLUMNS indicates that reliabilities are to be calculated on database columns.

MODULE OR GROUP ID

If the data type designation is ROWS, then a module ID must follow. If COLUMNS is designated, a group ID must follow. The module or group ID indicates the database rows or columns for which reliabilities are to be calculated.

FOR

The keyword FOR alerts the procedure to expect, depending on the type of the preceding data designation, a module or group ID.

GROUP OR MODULE ID

If a module ID occurs before the FOR keyword, then a group ID must follow. Conversely, if a group ID precedes the FOR keyword, then a module ID must follow. The group or module ID following the FOR keyword indicates the values across which reliabilities for rows or columns are to be calculated. Statistically, the group or module ID following the FOR keyword can be thought of as an indication of the number of observations contained in each of the rows or columns for which reliabilities are being calculated.

ADJUST

Specification of the optional keyword ADJUST indicates that, when calculating reliabilities, differences due to anchor points are not to be considered part of the error of measurement.

HEADING

The keyword HEADING indicates that the following character string enclosed in single quotes is to be used as a title on the printed output.

ASSIGNMENT OPERATOR

A "=" symbol. The assignment operator separates the HEADING keyword from the character string(s) serving as a report title.

CHARACTER STRING

Up to 10 lines of 131 characters each may comprise the title character string.

PERIOD

A period ('.') must end the syntax of the RELY procedure.

RELY EXAMPLES

EXAMPLE 1

```
BEGIN SAMPLEDATA80 EXECUTE.
ADDATA ROWS FOR INCUMBENTS N=6
    TRACTOR                    'OPERATE TRACTOR'
    JACKHAMMER                 'OPERATE JACKHAMMER'
    BULLDOZER                  'OPERATE BULLDOZER'
    POWERWRENCH                'OPERATE POWERWRENCH'
    FLAMETHROWER               'OPERATE FLAMETHROWER'
    TELEPHONE                  'OPERATE TELEPHONE'
    FORMAT '(7F1.0)'.
SELECT ROWS EQUIPMENT (TRACTOR JACKHAMMER BULLDOZER
    POWERWRENCH FLAMETHROWER TELEPHONE)
    'EQUIPMENT MODULE'.
ADDATA COLUMNS FOR EQUIPMENT N=4
    RATER1           'RATER NUMBER 1'
    RATER2           'RATER NUMBER 2'
    RATER3           'RATER NUMBER 3'
    RATER4           'RATER NUMBER 4'
    FORMAT '(8F1.0)'.
SELECT COLUMNS RATERS (RATER1 RATER2 RATER3 RATER4)
    'RATERS OF EQUIPMENT DIFFICULTY'.
RELY COLUMNS RATERS FOR EQUIPMENT
    HEADING='RELIABILITY OF EQUIPMENT DIFFICULTY RATINGS'.
END.
1100011
0010100
1100000
1001000
0000001
1000001
251726
473948
351968
362814
```

In the above example six rows are initially being added to the database that represent different equipment usage indices for each incumbent (1 if the incumbent operates the equipment and 0 if they do not). Each of the six equipment rows are being permanently appended to the database. The SELECT procedure following the first ADDATA command is forming the six equipment rows into a module named EQUIPMENT. The second ADDATA command is adding four columns to the database adjacent to the rows contained in module EQUIPMENT. The values of the four columns are from Winer (1971), page 288, and are being used in this example to represent equipment difficulty indices from four raters. The second SELECT command is forming the four columns of difficulty ratings into a group called RATERS. Finally, the RELY procedure is being invoked to calculate reliabilities on the four columns of rating measured across the six equipment rows.

EXAMPLE 1
PRINTED OUTPUT

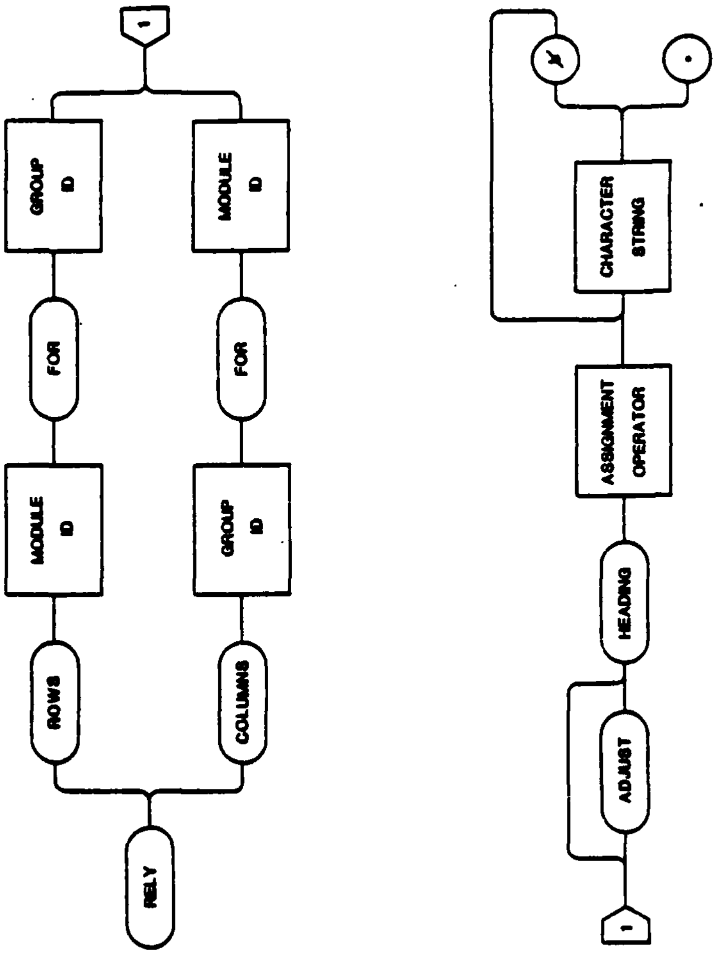
```
                                                                PAGE - 1

                          STUDY ID - SAMPLEDATA80
                  RELIABILITY OF EQUIPMENT DIFFICULTY RATINGS

RATERS - RATERS                     RATERS OF EQUIPMENT DIFFICULTY
TASKS  - EQUIPMENT                  EQUIPMENT MODULE

     R11      - 0.737705  RELIABILITY FOR A SINGLE RATER
     RKK      - 0.918367  RELIABILITY FOR THESE K RATERS
     BTSS     -  122.500  BETWEEN TASK SUM OF SQUARES
     WSS      -   36.000  WITHIN TASK SUM OF SQUARES
     BRSS     -   17.500  BETWEEN RATER SUM OF SQUARES
     RSS      -   18.500  RESIDUAL SUM OF SQUARES
     TSS      -  158.500  TOTAL SUM OF SQUARES
     BTMS     -   24.500  BETWEEN TASK MEAN SQUARE
     WMS      -    2.000  WITHIN TASK MEAN SQUARE
     BTMS     -    5.833  BETWEEN RATER MEAN SQUARE
     RMS      -    1.233  RESIDUAL MEAN SQUARE
     TMS      -    6.891  TOTAL MEAN SQUARE
     NRATER   -       4.  AVERAGE NUMBER OF TASKS
     N        -        6  NUMBER OF TASKS
     K        -        4  NUMBER OF RATERS

                                                                PAGE - 2

                          STUDY ID - SAMPLEDATA80
                  RELIABILITY OF EQUIPMENT DIFFICULTY RATINGS

RATER     RATER       NUMBER OF TASKS
NUMBER     ID      RATED BY THIS RATER    CORRELATION     T-VALUE
  1       RATER1            6.             0.986770       12.1729
  2       RATER2            6.             0.986772       12.1739
  3       RATER3            6.             0.793378        2.6067
  4       RATER4            6.             0.784837        2.5329
```

EXAMPLE 2

```
     BEGIN SAMPLEDATA80 EXECUTE.
     RELY COLUMNS RATERS FOR EQUIPMENT ADJUST
          HEADING='ADJUSTED RELY'.
     END.
```

The second example of the RELY procedure is requesting that reliabilities be calculated on exactly the same data as in the first example (since all data that was added was saved permanently, the second example is much simpler than the first). The main difference between the two RELY examples is that the second example is requesting that statistics be "adjusted" (anchor points are not to be considered part of the error of measurement). The output generated from the second RELY example will be very similar to that generated by the first RELY example, except that additional adjustment statistics will be printed (in this case RKK will equal .949660 and R11 will equal .825059).

RELY


RELY
ROWS
COLUMNS
MODULE ID
GROUP ID
FOR
FOR
GROUP ID
MODULE ID
1
1
ADJUST
HEADING
ASSIGNMENT OPERATOR
CHARACTER STRING
,
.

REPORT

INTRODUCTION

PCRPOSE

The REPORT procedure is used to facilitate documentation by producing an up-to-date listing of the information that resides on a given CODAP80 database.

FORM

The general form of the REPORT procedure is as follows:

1) The procedure keyword REPORT.
2) A keyword indicating the database information to be reported.
3) The optional keyword NOREMARKS.
4) A period or semicolon.

EXAMPLE

```
BEGIN SAMPLEDATA80 EXECUTE.
REPORT SYSCNST.
END.
```

The above REPORT syntax is requesting that a listing be produced displaying information on the system constants that reside on the database.

OUTPUT FROM PROCEDURE

As a function of user input, REPORT will produce a printed display of information pertaining to system constants, rows, columns, modules or groups that reside on a CODAP80 database.

REPORT SYNTAX

Refer to the syntax graph of the REPORT procedure.

REPORT

The keyword REPORT identifies the command.

ALL

Specification of the ALL keyword will produce a listing pertaining to all the stored information on the database.

MODULES

The appearance of the MODULES keyword in the REPORT procedure syntax will produce a listing of information pertaining to the system and created modules that reside on the database.

SYSMODS

Specification of the SYSMODS keyword will produce a listing of all database system modules.

CMODS

Specification of the CMODS keyword will produce a listing of all database created modules.

GROUPS

The appearance of the GROUPS keyword will produce a listing of information pertaining to the system and created groups that reside on the database.

SYSGROUPS

Specification of the SYSGROUPS keyword will generate a listing of information pertaining to the system groups that reside on the database.

CGRPS

A CGRPS keyword designation will result in a listing of the created groups on the database.

ROWS

The appearance of the ROWS keyword will produce a listing of both system and created rows on the database.

SYSROWS

A SYSROWS keyword designation will result in a listing of database system rows.

HROWS

The appearance of the HROWS keyword will generate a listing of the history rows on the database.

TROWS

The appearance of the TROWS keyword will generate a listing of the task rows on the database.

CROWS

A CROWS keyword designation will result in a listing of created database rows.

COLUMNS

The appearance of the COLUMNS keyword will generate a listing of both system and created columns on the database.

SYSCOLS

Specification of the SYSCOLS keyword will produce a listing of database system columns.

CCOLS

A CCOLS keyword designation will result in a listing of created database columns.

CONSTANTS

The appearance of the CONSTANTS keyword will produce a listing of system constants residing on the database.

SYSCNST

Specification of the keyword SYSCNST will produce a listing of database system constants.

NOREMARKS

Specification of the keyword NOREMARKS will suppress the printing of any remarks associated with the listed database information.

PERIOD OR SEMICOLON

A period must end the syntax of the REPORT procedure. For an illustration of the use of the terminating semicolon, see example 2 of REPORT.

REPORT EXAMPLES

EXAMPLE 1

```
BEGIN SAMPLEDATA80 EXECUTE.
REPORT TROWS.
END.
```

The above REPORT syntax will produce a listing of the task rows residing on the database.

EXAMPLE 1
PRINTED OUTPUT

```
                                                              PAGE - 1

                          STUDY ID - SAMPLEDATA80
                             TASK ROW REPORT

  ROW         REMARK
+-------+   +------------------------------------------------------+
  T  1        SUBDUE VIOLENT INMATES
  T  2        SHAKE DOWN INMATES
  T  3        SHAKE DOWN VISITORS
  T  4        ESCORT INMATES
  T  5        TESTIFY IN COURT
```

EXAMPLE 2

```
BEGIN SAMPLEDATA80 EXECUTE.
REPORT HROWS; SROWS.
END.
```

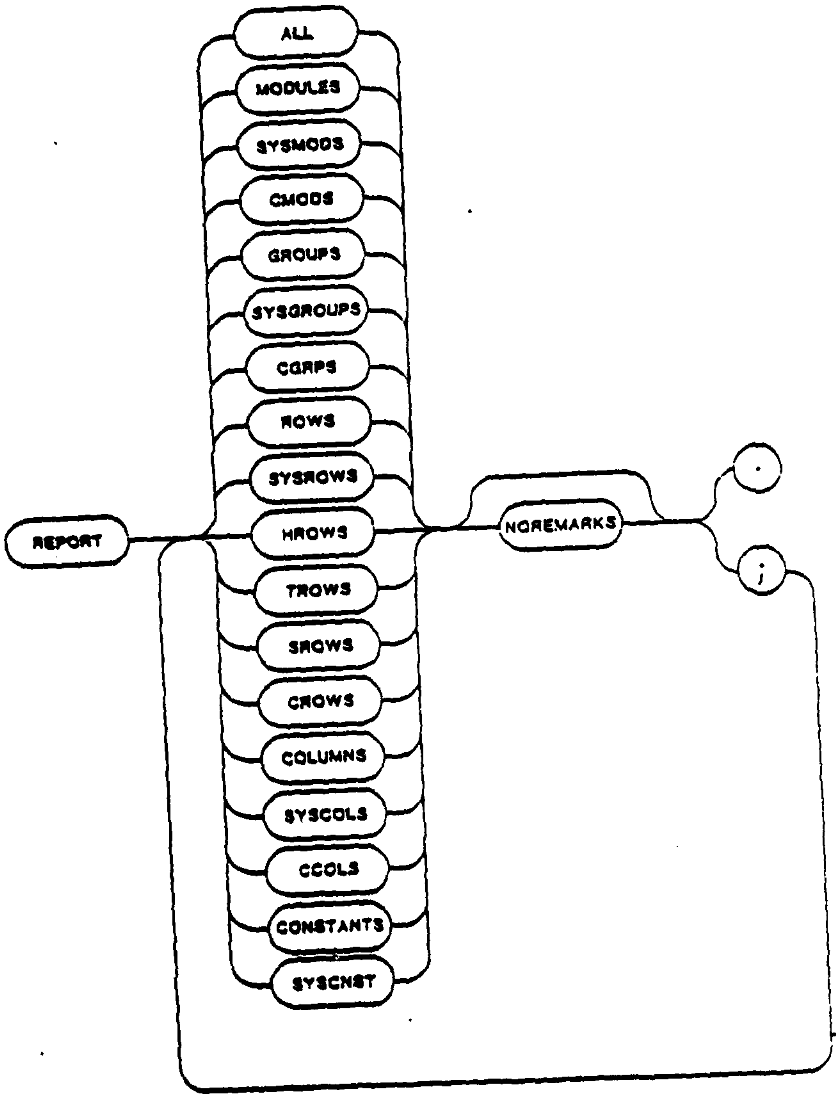
The above REPORT syntax will produce a listing of both the history and secondary rows residing on the database. Note the terminating semicolon.

EXAMPLE 2
PRINTED OUTPUT

```
                                                              PAGE - 1

                          STUDY ID - SAMPLEDATA80
                            HISTORY ROW REPORT

  ROW         REMARK
+-------+   +------------------------------------------------------+
  H  1        SEX
  H  2        AGE
  H  3        YEARS ON JOB
  H  4        INCUMBENT ID
```

EXAMPLE 2
PRINTED OUTPUT (continued)

```
                                                              PAGE - 2

                          STUDY ID - SAMPLEDATA80
                          SECONDARY ROW REPORT

  ROW        REMARK
+------+   +-----------------------------------------------------------+
  S  1       SECONDARY - SUBDUE VIOLENT INMATES
  S  2       SECONDARY - SHAKE DOWN INMATES
  S  3       SECONDARY - SHAKE DOWN VISITORS
  S  4       SECONDARY - ESCORT INMATES
  S  5       SECONDARY - TESTIFY IN COURT
```


REPORT
ALL
MODULES
SYSMODS
CMODS
GROUPS
SYSGROUPS
CGRPS
ROWS
SYSROWS
HROWS
TROWS
SROWS
CROWS
COLUMNS
SYSCOLS
CCOLS
CONSTANTS
SYSCNST
NOREMARKS
.
;

SELECT

INTRODUCTION

PURPOSE

The SELECT procedure defines aggregates of rows of columns on a database. SELECT provides the means by which CODAP80 users generate modules of database rows or groups of database columns that meet specified criteria. Generally, the CODAP80 user will not be interested in processing an entire database at one time but will only be concerned with a particular subset of the database. Through the use of the SELECT procedure aggregates of database rows or columns are assigned module or group ID's. Any future reference in other procedures to the selected ID alerts CODAP80 to direct processing to that subset of the database associated with it.

FORM

The general form of the SELECT procedure is as follows:

1) The procedure keyword SELECT.
2) A data type designation specifying whether rows or columns of the database are to be selected.
3) A user supplied valid CODAP80 ID that will be associated with the aggregate of database rows or columns selected.
4) Selection criteria defining which database rows or columns are to be members of the new module or group.
5) An indication of whether or not the new module or group ID is to be permanently saved for future reference.

EXAMPLE

```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT ROWS NEWMODULE (T2-T3) NOSAVE
    'SHAKE DOWN TASKS'.
END.
```

Execution of the above SELECT example will form a module (named NEWMODULE) of two tasks. The module will exist only for the duration of the computer run (as indicated by the NOSAVE keyword). The remark SHAKE DOWN TASKS will be associated with the module ID.

OUTPUT FROM PROCEDURE

Execution of the SELECT procedure produces no printed output. The result of executing the SELECT procedure will be a new module or group of database rows or columns being defined.

SELECT SYNTAX

Refer to the syntax graph of the SELECT procedure.

SELECT

The keyword SELECT identifies the command.

DATA TYPE DESIGNATION

The keyword ROWS or COLUMNS indicates whether a module or group is being selected.

ID

This is any valid 1-12 character CODAP80 ID supplied by the user. It will be associated with the module or group being selected.

COLUMN LIST

A column list is a list of database columns enclosed in parentheses. System columns and created columns may both be in the list. The columns appearing in the list will be included in the group being selected. A column ID appearing in the list may only be specified once. System columns appearing in the list must be specified in ascending numerical order. An example of a valid column list is "(I1-I3, I5)".

ROW LIST

A row list is a list of database rows enclosed in parentheses. System rows and created rows may both be in the list. The rows appearing in the list will be included in the module being selected. A row ID appearing in the list may only be specified once. System rows appearing in the list must be specified in ascending numerical order. An example of a valid row list is "(H1, T1-T3, S1, S5)".

BOOLEAN OPERATOR

A Boolean operator is used to connect a row list with a column Boolean expression or a column list with a row Boolean expression. Acceptable Boolean operators are ".AND." and ".OR." and help to define the criteria for module or group selection. If the Boolean operator is ".AND." it means only those elements of the preceding row or column list that meet the criteria of the following Boolean expression will be included in the group or module. If the Boolean operator is ".OR." it means all the elements of the preceding row or column list plus those that meet the criteria of the following Boolean expression will be included in the group or module. See SELECT example 7 for illustration.

ROW BOOLEAN EXPRESSION*

A row Boolean expression is a standard Boolean expression used to establish a set of criteria upon which to base the inclusion of a database column into a group. As an extension of the standard Boolean expression, selection criteria can be focused on a particular subset of database columns by defining that they be "IN" or "NOT IN" a specific group. See SELECT examples 3 and 4 for illustration.

COLUMN BOOLEAN EXPRESSION*

A column Boolean expression is a standard Boolean expression used to establish a set of criteria upon which to base the inclusion of a database row into a module. As an extension of the standard Boolean expression, selection criteria can be focused on a particular subset of database rows by defining that they be "IN" or "NOT IN" a specific module. See SELECT examples 5 and 6 for illustration.

NOSAVE

Specification of the optional keyword NOSAVE indicates that the defined group or module will not be permanently saved for future reference, but will exist only for the duration of the computer run.

REMARK

A remark is a string of up to 240 characters enclosed in single quotes. The remark will be associated with the group or module selected.

PERIOD OR SEMICOLON

A period ('.') must end the syntax of the SELECT procedure. If the syntax ends in a semicolon, another SELECT command may immediately follow without having to repeat the SELECT command keyword.

*NOTE

A Boolean expression may consist of relational operators, Boolean operators or both. Relational operators (often called comparison operators) propose a relationship between two quantities and ask CODAP80 to determine whether or not the relationship holds. The relational operators take the following form:

= or .EQ.	equal to
¬= or .NE.	not equal to
>= or .GE.	greater than or equal to
<= or .LE.	less than or equal to
> or .GT.	greater than
< or .LT.	less than

Boolean operators (often called logical infix operators) are usually used in expressions that also include relational operators. The Boolean operators take the following form:

& or .AND.
! or .OR.

See SELECT example 8 for an illustration of the use of both relational and Boolean operators in a Boolean expression.

SELECT EXAMPLES

EXAMPLE 1

```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT ROWS DUTYA (T1 T2 T4) 'INMATE TASKS';
       ROWS DUTYB (T3 T5) 'NON-INMATE TASKS';
       ROWS SHAKEDOWN (T2 T3) 'SHAKE DOWN TASKS';
       COLUMNS PEOPLE (I1-I3) 'FIRST 3 PEOPLE ON DATABASE'.
END.
```

The above SELECT syntax is generating three modules (DUTYA, DUTYB, and SHAKEDOWN) of database rows and one group (PEOPLE) of database columns. They will be permanently saved on the database for future reference. The SELECT syntax in example 1 illustrates the use of row lists and column lists to define the criterion for selection of a database row or column as a member of a module or group. Note that a database row may be selected for membership in more than one module (the same is true of database columns).

EXAMPLE 2

```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT COLUMNS MALES (H1=1) 'MALE INCUMBENTS';
       COLUMNS FEMALES (H1=2) 'FEMALE INCUMBENTS'.
END.
```

The SELECT syntax in example 2 illustrates the use of simple row Boolean expressions to define column membership in a group. The effect of the syntax in example 2 is to select those database columns in which H1=1 (H1 is sex; see Sample Database) as members of group MALES, and those in which H1=2 as members of group FEMALES. Group MALES will have the following columns as members: I2, I4-I7. Group FEMALES will have the columns I1 and I3 as members.

EXAMPLE 3

```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT COLUMNS OLDERMALES (H1=1 & H2 > 30)
       'OLDER MALE INCUMBENTS'.
END.
```

The SELECT command in example 3 is using a row Boolean expression to generate a group of database columns named OLDERMALES. Membership is defined as those database columns in which the rows H1 (Sex) equals 1 and H2 (Age) is greater than 30. The members of group OLDERMALES will be the database columns I4 and I6.

EXAMPLE 4

```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT COLUMNS FEMALE_G4 (H1=2 & IN G4)
        'FEMALE INCUMBENTS IN CLUSTER GROUP G4'.
END.
```

Example 4 of SELECT is demonstrating the use of a row Boolean expression to select a group (named FEMALE_G4) consisting of those incumbents who are female and also members of the system cluster group G4 (which was generated by the OGROUP database creation routine). The members of group FEMALE G4 will be the columns I1 and I3. Note the use of the "IN" parameter in the Boolean expression. Instead of specifying the system cluster group G4, the user could just have well specified the created group PEOPLE (which was selected in SELECT example 1). The effect would be the same.

EXAMPLE 5

```
BEGIN SAMPLEDATA80 EXECUTE.
DESCRIBE ROWS TASKS FOR (INCUMBENTS) PERCENT_RESP=PCNT
          'PERCENTAGE OF INCUMBENTS RESPONDING TO TASKS'.
SELECT ROWS HIPCNT_TASKS (PERCENT_RESP > 50)
          'TASKS WITH GREATER THAN 50% RESPONDING'.
END.
```

The CODAP80 syntax in SELECT example 5 is demonstrating how a user might go about selecting a module of task rows that had more than 50 percent of the incumbents responding. The DESCRIBE command is calculating, for every task row on the database, the percentage of incumbents responding. The effect of the command is to create a column of percentage values (named PERCENT_RESP) with a value for every task row:

	PERCENT_RESP
T1	71.43
T2	85.71
T3	71.43
T4	57.14
T5	42.86

The SELECT command in example 5 is using a column Boolean expression to select a module of those row elements of the created column PERCENT_RESP that exceed a value of 50. The selected module will be named HIPCNT_TASKS and will have the rows T1-T4 as members. To produce a listing of the task rows that were selected, the user need only reference the ID assigned to the module in,. say, the PRINT procedure, and CODAP80 will direct processing at the members in question.

EXAMPLE 6

```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT ROWS NEWMOD (PERCENT_RESP >=50 & PERCENT_RESP <=75
  & NOT IN DUTYB)
        'TASKS WITH 50%-75% PERFORMING AND NOT IN DUTYB'.
END.
```

The SELECT command in example 6 is creating a module (named NEWMOD) with the task rows T1 and T4 as members. The task rows selected correspond to those row elements of the column PERCENT_RESP that ranged in value from 50 to 75, and that were at the same time not members of module DUTYB (see example 1 and 5 of this procedure). Note the use of the "NOT IN" parameter in the column Boolean expression.

EXAMPLE 7

```
BEGIN SAMPLEDATA80 EXECUTE.
SELECT COLUMNS NEWGRP (I1-I5) .AND. (S2=1)
        'INMATE SHAKE DOWN ASSISTANT AMONG FIRST 5
          INCUMBENTS'.
END.
```

The SELECT command in example 7 is demonstrating how a column list and a row Boolean expression may be combined to define the criteria for group membership. The effect of the command is to create a group (named NEWGRP) with the columns I1, I2 and I5 as members. The appearance of the Boolean operator ".AND." between the column list and the Boolean expression defines the selection criteria for group membership as being only those elements of the preceding column list (I1-I5) that are true for the following row Boolean expression (S2=1). Had the Boolean operator between the column list and the row Boolean expression been ".OR." the selection criteria for group membership would have been those elements appearing in the preceding column list plus any columns that were true for the following row Boolean expression (resulting in a group with the columns I1-I6 as members).

EXAMPLE 8

```
BEGIN SAMPLEDATA80 EXECUTE.
DESCRIBE ROWS TASKS FOR (INCUMBENTS)
    PCNTRESPOND=PCNT 'PERCENT RESPONDING';
    NUMBRESPOND=N    'NUMBER RESPONDING'.
SELECT ROWS NEWMODULE (PCNTRESPOND > 70
    .AND. PCNTRESPOND < 90 .OR. NUMBRESPOND .EQ. 3)
    'MODULE MADE UP OF TASKS T1-T3 & T5'.
END.
```

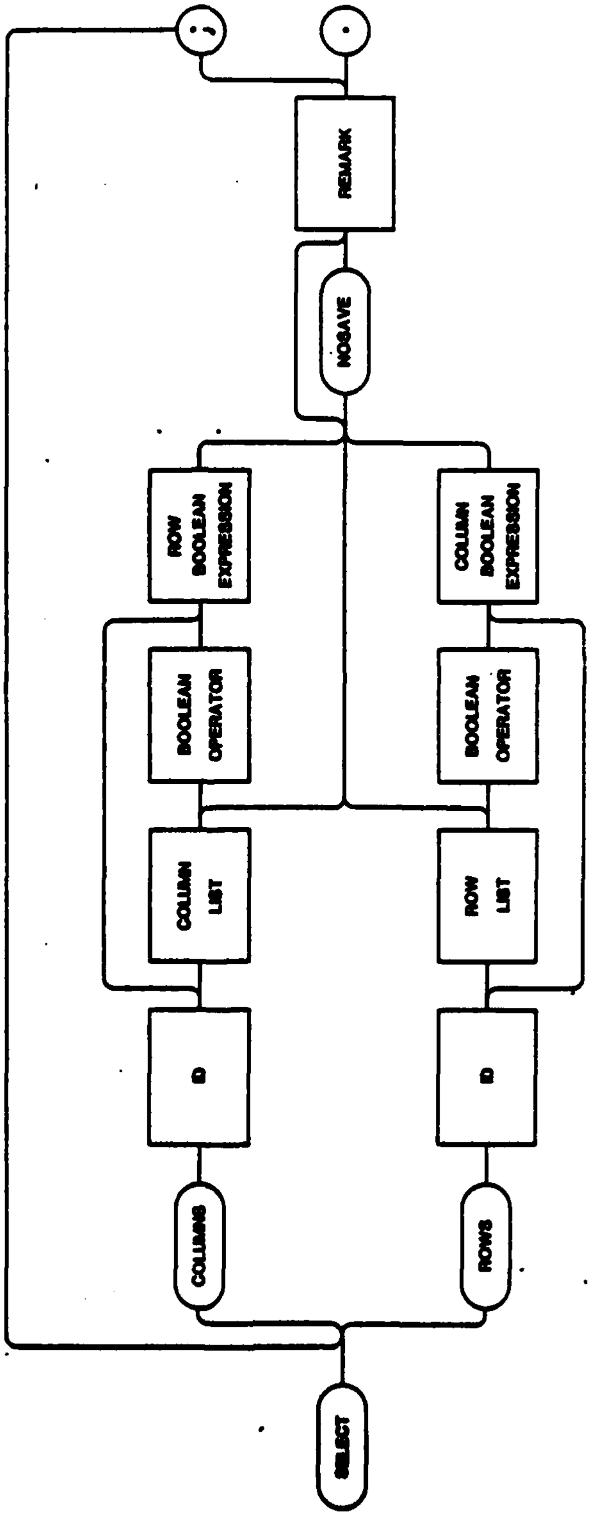
The DESCRIBE command in example 8 is generating two database columns. Column PCNTRESPOND will consist of the percentage of members responding to each task and column NUMBRESPOND will consist of the number of members responding to each task.

The values in the two columns will consist of:

	PCNTRESPOND	NUMBRESPOND
T1	71.43	5
T2	85.71	6
T3	71.43	5
T4	57.14	4
T5	42.86	3

The SELECT command in example 8 is using a column Boolean expression consisting of both relational operators and Boolean operators to select task rows into module NEWMODULE. The effect of the SELECT command is to select those task rows in which the column PCNTRESPOND is greater than 70 and less than 90 (T1, T2 and T3) or the column NUMBRESPOND is equal to 3 (T5). Based on the criteria defined in the SELECT command the selected rows will be T1-T3 and T5.

SELECT


SELECT
COLUMNS
ROWS
ID
ID
COLUMN LIST
ROW LIST
BOOLEAN OPERATOR
BOOLEAN OPERATOR
ROW BOOLEAN EXPRESSION
COLUMN BOOLEAN EXPRESSION
NOSAVE
REMARK
;
.

STANDARD

INTRODUCTION

PURPOSE

The STANDARD command standardizes specified rows or columns of the database to any given mean and standard deviation. For each row or column specified, STANDARD will create a new standardized row or column and, if indicated, store it permanently on the database.

FORM

The general form of the STANDARD command is as follows:

1) The procedure keyword STANDARD.
2) The keyword ROWS or COLUMNS - this keyword alerts the system that either rows or columns of the database are to be standardized.
3) A description of which rows or columns of the database are to be standardized.
4) A group or module designation representing the "length" or the number of elements that are contained in the row(s) or column(s) that is being standardized.
5) A user-supplied constant indicating the mean the standardized values are to take.
6) A user-supplied constant indicating the standard deviation the standardized values are to take.
7) A new ID. The new ID will have a numeric value, ranging from 1 to the number of rows or columns specified in 3, appended to it by the system. The user must be careful not to specify an ID that will conflict with one previously defined in the database. The user must also take care to specify an ID that, when the numeric value is appended to it by the system, is not longer than 12 characters. If only a single row or column is being standardized, a numeric value is not appended to the new ID.
9) Optionally, the keyword NOSAVE. If NOSAVE is specified, then the new IDs created for this run will not be retained for future use.
10) A remark that will be associated with the new IDs.

EXAMPLE

```
STANDARD ROWS (H1) FOR INCUMBENTS
     MEAN:=50 STD:=10 NEWROW 'NEW STANDARDIZED ROW'.
```

The above STANDARD command syntax is requesting that the database row H1 be standardized to a mean of 50 and a standard deviation of 10 for every incumbent on the database. The standardized values of H1 will be named NEWROW and this created vector will be permanently added as a new row on the database.

OUTPUT FROM PROCEDURE

Execution of the STANDARD command produces no printed output.

STANDARDIZATION FORMULA

The equation used by STANDARD for standardization is:

$$T = \frac{S'(X - \overline{X})}{S} X'$$

Where:

T = Standardized value.
X = Original raw data point.
$\overline{X}$ = Original mean of row/column being standardized.
S = Original standard deviation of row/column being standardized.
X' = User specified constant indicating the mean the new standardized row/column will take.
S' = User specified constant indicating the standard deviation the new standardized row/column will take.

STANDARD SYNTAX

Refer to the syntax graph of the STANDARD procedure.

STANDARD

The keyword STANDARD identifies the command.

DATA TYPE DESIGNATION

The keyword ROWS or COLUMNS indicates whether rows or columns of the database are to be standardized.

MODULE ROW LIST

A Module Row List (MROWLT) is a list of at least one module or row ID enclosed in parentheses. Lists of module IDs, system row lists and lists of row IDs may all occur together in a MROWLT. If the data type designation following the STANDARD command keyword is ROWS, then a MROWLT must follow. The MROWLT serves to indicate to the STANDARD procedure which rows of the database are to be standardized.

CAUTION: All created module IDs appearing in the MROWLT must have been selected and permanently saved during a previous execution of the CODAP80 interpreter. STANDARD cannot process created modules that were selected in the same run stream.

GROUP COLUMN LIST

A Group Column List (GCOLST) is a list of at least one group or column ID enclosed in parentheses. Lists of group IDs, system column lists and lists of column IDs may all occur together in a GCOLST. If the data type designation following the standard command keyword is COLUMNS, then a GCOLST must follow. The GCOLST serves to indicate to the STANDARD procedure which columns of the database are to be standardized.

CAUTION: All created group IDs appearing in the GCOLST must have been selected and permanently saved during a previous execution of the CODAP80 interpreter. STANDARD cannot process created groups that were selected in the same run stream.

FOR

The FOR keyword alerts the STANDARD procedure to expect a following group or module ID.

GROUP ID

A group ID is an identified aggregate of database columns. A group ID following the FOR keyword indicates the columns of the database the rows are to be standardized across. If the preceeding data type designation was ROWS, than a group ID must follow the FOR keyword.

MODULE ID

A module ID is an identified aggregate of database rows. A module ID following the FOR keyword indicates the rows of the database the columns are to be standardized across. If the preceeding data type designation was COLUMNS, then a moduie ID must follow the FOR keyword.

MEAN

The keyword MEAN serves to alert STANDARD that the following user supplied constant represents the mean to which the rows or columns are to be standardized.

STD

The keyword STD serves to alert STANDARD that the following user supplied constant represents the standard deviation to which the rows or columns are to be standardized.

ASSIGNMENT OPERATOR

Either the symbols '=' or ':='. Either of these symbols may be used to separate the MEAN or STD keywords from their associated user supplied constant.

CONSTANT

A user supplied numeric value, such as '3.14'.

ID

Any valid CODAP80 ID, supplied by the user. This new ID will have a number value, ranging from 1 to the number of rows or columns specified in the MROWLT or GCOLST, appended to it by the system. If only a single row or column is being standardized, then a numeric value is <u>not</u> appended to the ID.

NOSAVE

If the keyword NOSAVE is specified, any new IDs created will not be saved for future reference.

REMARK

This is a string of up to 240 characters, enclosed in single quotes. The remark will be associated with the new IDs created. A remark must be associated with the new IDs.

PERIOD

A period ('.') must end the STANDARD statement.

STANDARD EXAMPLES

EXAMPLE 1

```
STANDARD COLUMNS (G6) FOR SVARS
    MEAN:=50 STD:=10 STANI
    'INCUMBENT COLUMN STANDARDIZED (M=50 S=10) FOR SVARS'.
```

The above STANDARD statement syntax is requesting that each of the columns defined by the system group ID G6 (G6 is a system cluster group ID defined by the OGROUP routine when the incumbents were clustered at database creation time), which is, referring to the Sample Database, every incumbent column, be standardized to a mean of 50 and a standard deviation of 10 across the rows defined by the system module ID SVARS (S1-S5). Seven new columns will be added to the database (one for each of the seven incumbent columns) and will be named STANI1-STANI7. The remark INCUMBENT COLUMN STANDARDIZED (M=50 STD=10) FOR SVARS will be associated with each of the new columns.

Referring to the Sample Database, the data to be standardized consists of:

	I1	I2	I3	I4	I5	I6	I7
S1	.	.	.	2	.	2	.
S2	1	1	.	2	1	1	3
S3	1	2	2	.	1	.	3
S4	2	1	2	2	.	.	.
S5	.	.	1	1	3	.	.

After standardization, the new columns consist of:

	STANI1	STANI2	STANI3	STANI4	STANI5	STANI6	STANI7
S1	.	.	.	55.00	.	57.07	.
S2	44.23	44.23	.	55.00	44.23	42.93	50.00
S3	44.23	61.55	55.77	.	44.23	.	50.00
S4	61.55	44.25	55.77	55.00	.	.	.
S5	.	.	38.45	35.00	61.55	.	.

EXAMPLE 2

```
STANDARD ROWS (H2) FOR INCUMBENTS
     MEAN:=50 STD:=10 H2STAN
     'ROW H2 STANDARDIZED (M=50 S=10) FOR INCUMBENTS'.
```

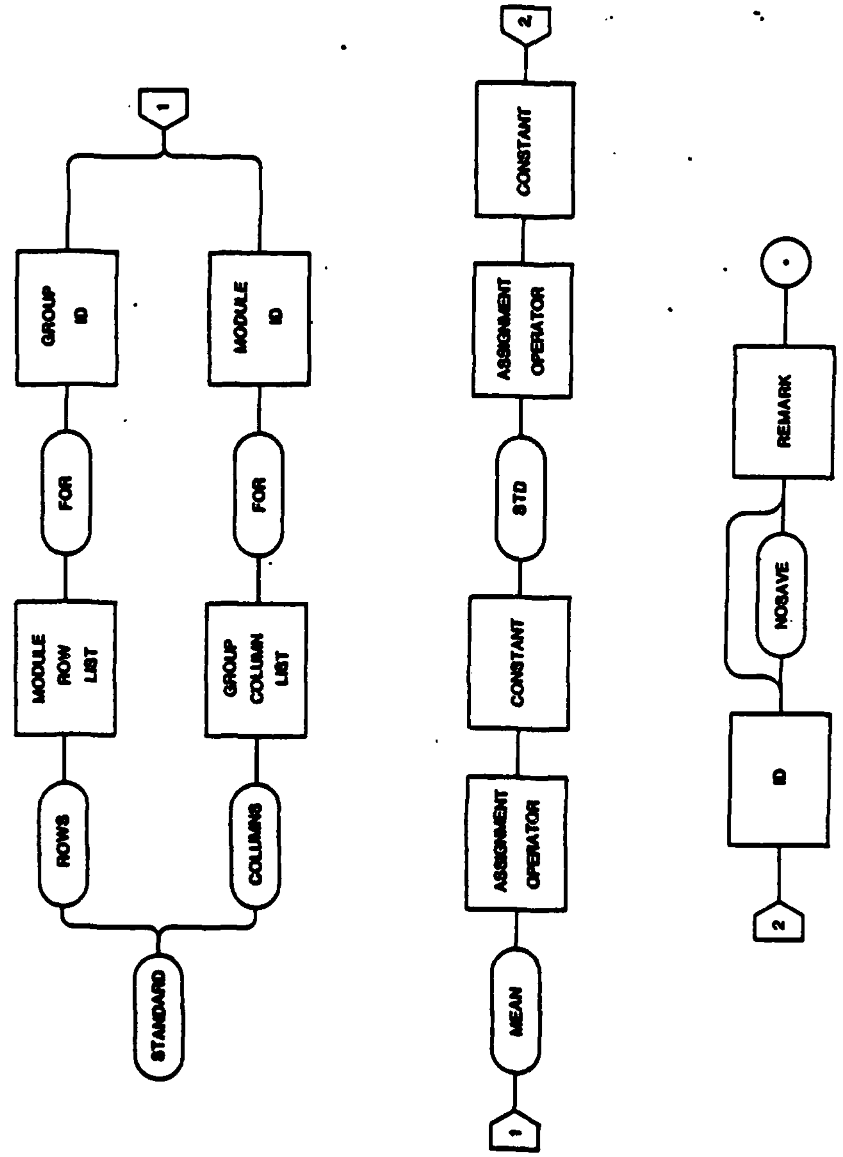
The above STANDARD statement syntax is requesting that the database row H2 be standardized to a mean of 50 and a standard deviation of 10 across every incumbent column in the database. The standardized row will be named H2STAN and will be permanently stored on the database along with its associated remark ROW H2 STANDARDIZED (M=50 S=10) FOR INCUMBENTS.

Referring to the Sample Database, the data to be standardized consists of:

	I1	I2	I3	I4	I5	I6	I7
H2	19	23	.	41	27	53	.

After standardization, the new row consists of:

	I1	I2	I3	I4	I5	I6	I7
H2STAN	40.35	43.19	.	55.96	46.03	67.47	.


STANDARD
ROWS
MODULE ROW LIST
FOR
GROUP ID
COLUMNS
GROUP COLUMN LIST
FOR
MODULE ID
1
1
MEAN
ASSIGNMENT OPERATOR
CONSTANT
STD
ASSIGNMENT OPERATOR
CONSTANT
2
2
ID
NOSAVE
REMARK
•

VARSUM

INTRODUCTION

PURPOSE

The VARSUM procedure produces frequency counts and percentages of the distribution of values for specified rows or columns on the database. The VARSUM procedure is particularly useful when comparing the distribution of a specified history variable across groups of interest generated from a cluster operation.

FORM

The general form of the VARSUM command is as follows:

1) The procedure keyword VARSUM.
2) The data type designation ROWS or COLUMNS.
3) A description of the rows or columns upon which distribution statistics are to be calculated.
4) A description of the aggregate of rows or columns (a group or module ID) across which distribution statistics are to be calculated.
5) Options controlling the type of distribution statistic calculated (frequencies or percentages - or both) and the appearance of the output.

EXAMPLE

```
VARSUM ROWS (S1) FOR (G6) COUNT
HEADING:='SIMPLE EXAMPLE OF THE VARSUM PROCEDURE'.
```

The above example VARSUM command syntax will answer the question, "What is the frequency distribution of the values of the secondary variable S1 across those incumbents identified by the system group G6?"

OUTPUT FROM PROCEDURE

The VARSUM procedure produces a report showing frequency counts or percentages (or both) of the distribution of values for specified rows or columns of the database.

VARSUM SYNTAX

Refer to the syntax graph of the VARSUM procedure.

VARSUM

The keyword VARSUM identifies the command.

DATA TYPE DESIGNATION

The keyword ROWS or COLUMNS designates whether it is to be rows or columns of the database upon which distribution statistics are to be calculated.

MODULE ROW LIST

A Module Row List (MROWLT) is a list of at least one module or row ID enclosed in parentheses. Lists of module IDs, system row lists and lists of row IDs may all appear together in a MROWLT. In regard to the VARSUM procedure, the MROWLT identifies the rows of the database upon which distribution statistics are to be calculated. A MROWLT must be specified if the data type designation was ROWS.

GROUP COLUMN LIST

A Group Column List (GCOLST) is a list of at least one group or column ID enclosed in parentheses. Lists of group IDs, system group lists, system column lists and lists of column IDs may all appear together in a GCOLST. In regard to the VARSUM procedure, the GCOLST identifies the columns of the database upon which distribution statistics are to be calculated. A GCOLST must be specified if the data type designation was COLUMNS.

FOR

The keyword FOR alerts the procedure that the following list of database row or column aggregates (that is, a list of module or group IDs) represent that part of the database across which distribution statistics are to be calculated. If the data type designation was ROWS, then a group list must follow the FOR keyword. If it was COLUMNS, then a module list must follow the FOR keyword.

MODULE LIST

A Module List is a list of at least one module ID enclosed in parentheses. Each module ID appearing in the module list identifies the rows of the database across which column distribution statistics are to be calculated.

GROUP LIST

A Group List is a list of at least one group ID enclosed in parentheses. Each group ID appearing in the group list identifies the columns of the database across which row distribution statistics are to be calculated.

COUNT

Specifying the keyword COUNT signifies that the distribution statistics calculated are to consist of frequency counts.

PERCENT

Specifying the keyword PERCENT signifies that the distribution statistics calculated are to consist of percentages.

DECODE

At the time the database was initially created (through the use of the INPSTD database creation routine) the user had the option of associating descriptive text with the values of a specified row. For example, the user could have associated the label 'MALE' with a sex value of '1' and 'FEMALE' with a sex value of '2'. If the user specifies DECODE in the VARSUM syntax, the procedure will substitute the associated label for the values of the ID for which distribution statistics are being calculated (see VARSUM example 2). The number of row or column aggregates across which distribution statistics were calculated that can be displayed across a page of output is eight. If DECODE is specified, only six row or column aggregates can be displayed across a page.

MISSING

The default condition of the VARSUM procedure is not to accumulate distribution statistics on missing values. If the MISSING keyword is specified, distribution statistics including missing values will be generated.

STAT

Specification of the keyword STAT indicates that mean and standard deviation statistics are to be calculated and printed along with the distribution statistics.

HEADING

The keyword HEADING indicates that the following character string(s) enclosed in single quotes is to serve as a report title.

ASSIGNMENT OPERATOR

Either of the symbols '=' or ':='. Either of these symbols may be used to separate the HEADING keyword from the character strings serving as a report title.

CHARACTER STRING

Up to 10 lines of 131 characters each may comprise the character strings serving as a report title. Each string of up to 131 characters (representing one title line) must be enclosed in single quotes. The beginning of a new title line is indicated by a blank and another title line enclosed in quotes.

For example:

```
HEADING:='EXAMPLE SHOWING HOW'  'REPORT TITLE LINES'
'ARE CONSTRUCTED FOR THE VARSUM PROCEDURE'.
```

This example would produce three title lines centered at the top of VARSUM's output page:

```
          EXAMPLE SHOWING HOW
           REPORT TITLE LINES
ARE CONSTRUCTED FOR THE VARSUM PROCEDURE
```

PERIOD

A period ('.') must end the syntax of the VARSUM procedure.

VARSUM EXAMPLES

EXAMPLE 1

```
BEGIN SAMPLEDATA80 EXECUTE.
VARSUM ROWS (S3,S4) FOR (G6) COUNT
  HEADING:='VARSUM EXAMPLE 1'
            'DISTRIBUTION OF S3 & S4 ACROSS ALL INCUMBENTS'.
END.
```

In the above example, the user is requesting that frequency counts be calculated on the distribution of values occurring for the rows S3 and S4 across all columns indicated by the system group G6 (G6 is a system group generated by clustering at database creation time. G6 contains 7 members: I1-I7).

EXAMPLE 1
PRINTED OUTPUT

```
                                                        PAGE - 1

                        STUDY ID - SAMPLEDATA80
                          VARSUM EXAMPLE 1
                DISTRIBUTION OF S3 & S4 ACROSS ALL INCUMBENTS

                          **** FREQUENCY ****

S - 3          SECONDARY - SHAKE DOWN VISITORS
-----------
INTERVAL                       G - 6
-----------              -----------
        1.00                       2
        2.00                       2
        3.00                       1
-----------              -----------
TOTAL COUNTED ABOVE                5
MISSING                            2

S - 4          SECONDARY - ESCORT INMATES
-----------
INTERVAL                       G - 6
-----------              -----------
        1.00                       1
        2.00                       3
-----------              -----------
TOTAL COUNTED ABOVE                4
MISSING                            3
```

EXAMPLE 2

```
     BEGIN SAMPLEDATA80 EXECUTE.
     SELECT ROWS NEWMOD (S3, S4)
        'MODULE CONTAINING ROWS S3 & S4'.
     VARSUM ROWS (NEWMOD) FOR (G5,G6)
        COUNT PERCENT DECODE MISSING
        HEADING:='VARSUM EXAMPLE 2'
        'DISTRIBUTION OF EACH ROW CONTAINED IN MODULE NEWMOD'
        'ACROSS COLUMNS IDENTIFIED BY SYSTEM GROUP G5'
        'AND THEN ACROSS THOSE COLUMNS IN SYSTEM GROUP G6'.
     END.
```

In the above example, the user is first selecting the rows S3 and S4 to be in the created module NEWMOD (see the section on the SELECT procedure for more information). Following that, the user is requesting that the VARSUM procedure calculate both frequency and percentage statistics for each row identified by the module ID NEWMOD (rows S3 and S4). The statistics are to be calculated first across the columns identified by the system cluster group G5, and then across the columns identified by the system cluster group G6 (G5 and G6 are system groups generated by clustering at database creation time. Referring to the Sample Database, G5 contains 4 members: columns I4-I7; G6 contains 7 members: columns I1-I7). Decode has been specified and missing values are to be included in the calculation of distribution statistics.

EXAMPLE 2
PRINTED OUTPUT

```
                                                              PAGE - 1

                         STUDYID - SAMPLEDATA80
                          VARSUM EXAMPLE 2
          DISTRIBUTION OF EACH ROW CONTAINED IN MODULE NEWMOD
             ACROSS COLUMNS IDENTIFIED BY SYSTEM GROUP G5
           AND THEN ACROSS THOSE COLUMNS IN SYSTEM GROUP G6

                         **** FREQUENCY ****

S - 3                  SECONDARY - SHAKE DOWN VISITORS
---------------
INTERVAL                            G - 5               G - 6
---------------          ---------------     ---------------
         .                              2                   2
     1.00 DO                            1                   2
     2.00 ASSIST                        0                   2
     3.00 SUPERVISE                     1                   1
---------------          ---------------     ---------------
TOTAL COUNTED ABOVE                     4                   7

                        **** PERCENTAGE ****

S - 3                  SECONDARY - SHAKE DOWN VISITORS
---------------
INTERVAL                            G - 5               G - 6
---------------          ---------------     ---------------
         .                          50.00               28.57
     1.00 DO                        25.00               28.57
     2.00 ASSIST                     0.00               28.57
     3.00 SUPERVISE                 25.00               14.29
---------------          ---------------     ---------------
TOTAL PERCENT                      100.00              100.00
```

EXAMPLE 2
PRINTED OUTPUT (continued)

```
                                                                  PAGE - 2

                         STUDYID - SAMPLEDATA80
                           VARSUM EXAMPLE 2
          DISTRIBUTION OF EACH ROW CONTAINED IN MODULE NEWMOD
             ACROSS COLUMNS IDENTIFIED BY SYSTEM GROUP G5
           AND THEN ACROSS THOSE COLUMNS IN SYSTEM GROUP G6

                           **** FREQUENCY ****

          S - 4           SECONDARY - ESCORT INMATES
          -------------
          INTERVAL                     G - 5            G - 6
          -------------        -------------    -------------
               .                           3                3
               1.00 DO                     0                1
               2.00 ASSIST                 1                3
          -------------        -------------    -------------
          TOTAL COUNTED ABOVE              4                7

                           **** PERCENTAGE ****

          S - 4           SECONDARY - ESCORT INMATES
          -------------
          INTERVAL                     G - 5            G - 6
          -------------        -------------    -------------
               .                       75.00            42.86
               1.00 DO                  0.00            14.29
               2.00 ASSIST             25.00            42.86
          -------------        -------------    -------------
          TOTAL PERCENT               100.00           100.00
```

EXAMPLE 3

```
     BEGIN SAMPLEDATA80 EXECUTE.
     VARSUM COLUMNS (G1) FOR (SVARS) COUNT
        HEADING:='VARSUM EXAMPLE 3'
                 'DISTRIBUTION OF EACH COLUMN CONTAINED IN'
                 'SYSTEM GROUP G1'
                 'ACROSS ROWS IDENTIFIED BY SYSTEM MODULE SVARS'.
     END.
```

The above example demonstrates the VARSUM procedure's symmetric capability. The two previous examples of the VARSUM procedure were calculating distribution statistics on rows across columns. Example 3 is requesting that distribution statistics be calculated on database columns extending across rows. Specifically, the user is requesting that frequency counts of the distribution of values for each of the columns contained in system group G1 (columns I2 and I3) be calculated across the rows identified by the system module SVARS (rows S1-S5).

EXAMPLE 3
PRINTED OUTPUT

```
                                                              PAGE - 1

                          STUDY ID - SAMPLEDATA80
                            VARSUM EXAMPLE 3
                  DISTRIBUTION OF EACH COLUMN CONTAINED IN
                             SYSTEM GROUP G1
               ACROSS ROWS IDENTIFIED BY SYSTEM MODULE SVARS

                           **** FREQUENCY ****

     I - 2
     ---------------
     INTERVAL                          SVARS
     ---------------          ---------------
                1.0                        2
                2.0                        1
     ---------------          ---------------
     TOTAL COUNTED ABOVE                   3
     MISSING                               2

     I - 3
     ---------------
     INTERVAL                          SVARS
     ---------------          ---------------
                1.0                        1
                2.0                        2
     ---------------          ---------------
     TOTAL COUNTED ABOVE                   3
     MISSING                               2
```

EXAMPLE 4

```
BEGIN SAMPLEDATA80 EXECUTE.
DESCRIBE ROWS TASKS FOR (G6)
  G6PCNT:=PCNT
  'PERCENT PERFORMING TASKS -- G6'.
CREATE COLUMN FOR TASKS
  IF G6PCNT  .LE. 60 THEN NEWCOLUMN:=1.0
  ELSE NEWCOLUMN:=2.0
  'G6PCNT <= 60, NEWCOLUMN=1 -- ELSE NEWCOLUMN=2'.
VARSUM COLUMNS (NEWCOLUMN) FOR (TASKS) COUNT
  HEADING:='VARSUM EXAMPLE 4'
             'DISTRIBUTION OF THE COLUMN NEWCOLUMN'
             'AS MEASURED ACROSS THE SYSTEM MODULE TASKS'.
END.
```

The above example is demonstrating how other procedures in CODAP80 may be used to add rows or columns of summary calculations to the database, and then have the VARSUM procedure produce a report of the distribution of those rows or columns.

Initially, the user is requesting that the DESCRIBE procedure generate a column consisting of the percent of all incumbents performing each task row. The column generated by DESCRIBE (and named G6PCNT) will be 5 elements long (one element per task) and will consist of the values:

```
71.43     85.71     71.43     57.14     42.86
```

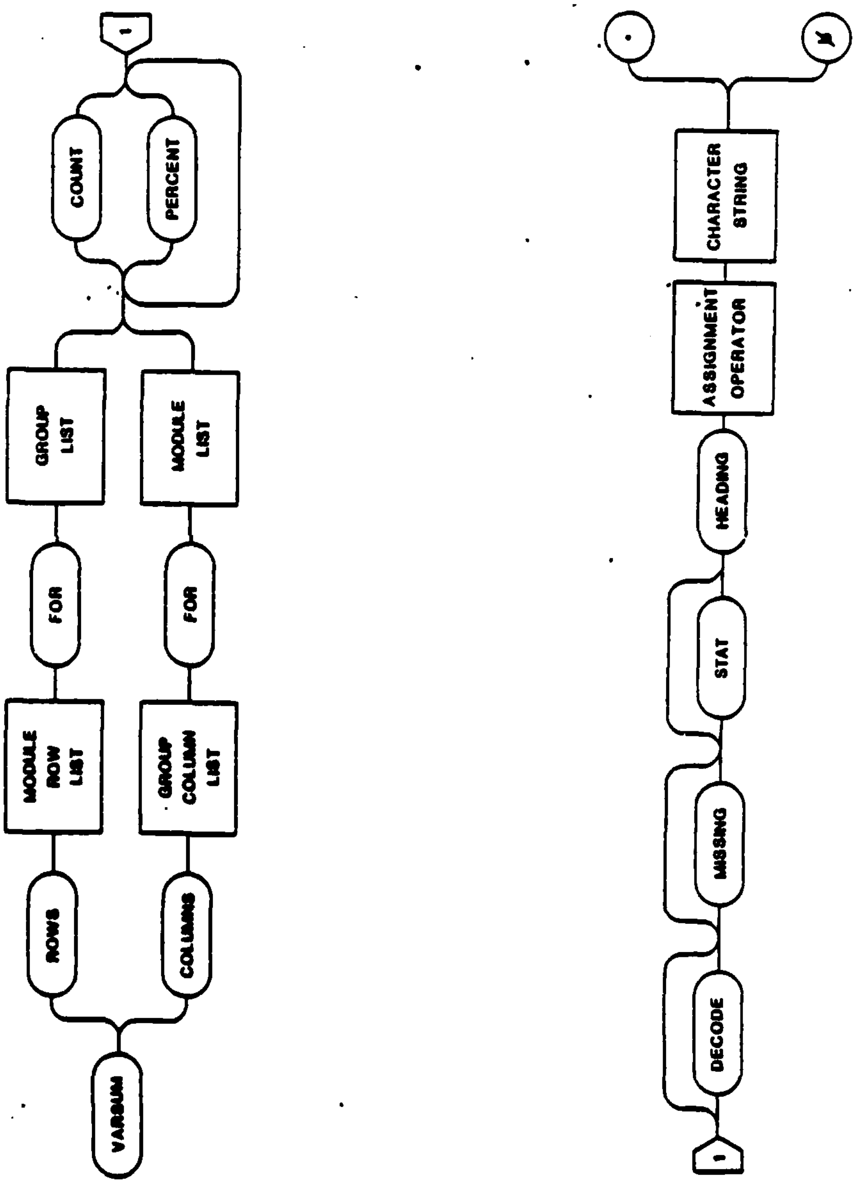
Following that, the user is requesting that the CREATE procedure generate another column (named NEWCOLUMN), the values of which to be a function of the magnitude of the values in column G6PCNT (NEWCOLUMN will equal 1.00 when G6PCNT is less than or equal to 60, otherwise NEWCOLUMN will equal 2.00). The column NEWCOLUMN will be 5 elements long (one for each task row) and will consist of the values:

```
2.00     2.00     2.00     1.00     1.00
```

Last, the user is requesting that the VARSUM procedure calculate frequency counts of the distribution of values in column NEWCOLUMN as measured across the rows identified by the system module TASKS (T1-T5).

EXAMPLE 4
PRINTED OUTPUT

```
                                                                 PAGE - 1

                          STUDY ID - SAMPLEDATA80
                            VARSUM EXAMPLE 4
                   DISTRIBUTION OF THE COLUMN NEWCOLUMN
                AS MEASURED ACROSS THE SYSTEM MODULE TASKS

                           **** FREQUENCY ****

   NEWCOLUMN      G6PCNT <= 60, NEWCOLUMN=1 -- ELSE NEWCOLUMN=2
   -------------
   INTERVAL                         TASKS
   -------------             -------------
         1.0                            2
         2.0                            3
   -------------             -------------
   TOTAL COUNTED ABOVE                  5
   MISSING                              0
```


VARSUM
ROWS
COLUMNS
MODULE ROW LIST
GROUP COLUMN LIST
FOR
FOR
GROUP LIST
MODULE LIST
COUNT
PERCENT
1
1
DECODE
MISSING
STAT
HEADING
ASSIGNMENT OPERATOR
CHARACTER STRING
.
ø

REFERENCES

REFERENCES

Brown, Gary D. System 370 Job Control Language. New York: John Wiley & Sons, 1977.

Ward, J.H., Jr. Hierarchical grouping to optimize an objective funotion. American Statistical Association Journal, 1963, 58, 236-244.

Winer, B.J. Statistical Principles in Experimental Design. New York: McGraw-Hill, 1971.

APPENDIX A

SAMPLE CODAP80 PROGRAM

```
    # ------------------------------------------------------------- #
    # THIS IS AN EXAMPLE OF A COMPLETE PROGRAM RUN STREAM            #
    # IN THE CODAP80 LANGUAGE.  A PROGRAM SUCH AS THIS               #
    # WOULD BE SUBMITTED TO THE COMPUTER BY THE USER.  A             #
    # GOOD PRACTICE TO FOLLOW WHEN WRITING CODAP80 SOURCE            #
    # PROGRAMS IS TO DOCUMENT WHAT THE PROGRAM IS DOING              #
    # THROUGH THE LIBERAL USE OF COMMENTS. ANY CHARACTER             #
    # STRING OCCURRING BETWEEN TWO POUND SIGNS IS                    #
    # INTERPRETED BY THE CODAP80 SYSTEM AS A COMMENT.                #
    # COMMENTS ARE NOT EXECUTED, BUT ARE PRINTED OUT                 #
    # ALONG WITH THE PROGRAM STATEMENTS.  FUTURE USERS               #
    # WILL THEN BE ABLE TO LOOK AT THE PROGRAM AND TELL              #
    # WHAT IT WAS DOING.                                             #
    # ------------------------------------------------------------- #
    # ------------------------------------------------------------- #
    # THE BEGIN STATEMENT IS THE FIRST EXECUTABLE                    #
    # STATEMENT IN THE CODAP80 LANGUAGE. THIS STATEMENT              #
    # ALERTS THE SYSTEM THAT A CODAP80 SOURCE LANGUAGE               #
    # PROGRAM FOLLOWS.  SAMPLEDATA80 IS THE STUDY ID                 #
    # ASSOCIATED WITH THE DATABASE BEING ACCESSED.  THE              #
    # STUDY ID GIVEN IN THIS STATEMENT WILL BE CHECKED               #
    # AGAINST THE ONE STORED ON THE DATABASE (WHICH WAS              #
    # ASSIGNED AT INPUT STANDARD TIME) AND, IF THEY                  #
    # MATCH, PROCESSING WILL CONTINUE.  THE KEYWORD                  #
    # EXECUTE INSTRUCTS THE SYSTEM THAT IF NO ERRORS                 #
    # ARE FOUND THE FOLLOWING STATEMENTS ARE TO BE                   #
    # EXECUTED.  HAD "EXECUTE" BEEN OMITTED, ONLY SYNTAX             #
    # ANALYSIS WOULD HAVE BEEN PERFORMED.                            #
    # ------------------------------------------------------------- #
BEGIN SAMPLEDATA80 EXECUTE.
    # ------------------------------------------------------------- #
    # ONE OF THE FIRST OPERATIONS IN A STUDY IS TO DEFINE            #
    # THE DATABASE SUBSETS OF INTEREST.  THE FOLLOWING               #
    # FIVE SELECT STATEMENTS ARE ASSIGNING TASKS (ROWS)              #
    # TO MODULES (DUTYA DUTYB) AND INCUMBENTS (COLUMNS)              #
    # TO GROUPS (MALES, FEMALES AND OLDERNOTING2).  THE              #
    # EFFECT OF THE FIFTH SELECT STATEMENT:                          #
    #                                                                #
    # COLUMNS OLDERNOTING2 (H2.GT.30 & NOT IN G2)                    #
    #   'INCUMBENTS OLDER THAN 30 AND NOT IN CLUSTER G2'             #
    #                                                                #
    # IS TO ASSIGN ONE INCUMBENT (I6) TO GROUP ID                    #
    # OLDERNOTING2.  THIS INCUMBENT (I6) IS THE ONLY ONE             #
    # IN THE SAMPLE DATABASE THAT MEETS THE CONDITION OF             #
    # BEING OLDER THAN 30 (HISTORY VARIABLE 2 IS AGE--SEE            #
    # SAMPLE DATABASE) WHILE AT THE SAME TIME NOT                    #
    # BELONGING TO CLUSTER G2.  THE REMARK 'INCUMBENTS               #
    # OLDER THAN 30 AND NOT IN CLUSTER G2' WILL BE STORED            #
    # ON THE DATABASE ALONG WITH ITS ASSOCIATED GROUP ID             #
    # (OLDERNOTING2) FOR LATER REFERENCE.                            #
```

```
        #  NOTICE THAT IT WAS NOT NECESSARY TO REPEAT THE  #
        #  SELECT PROCEDURE KEYWORD BECAUSE THE FIVE COMMANDS #
        #  OCCUR TOGETHER AND, EXCEPT FOR THE LAST IN THE  #
        #  SERIES, ARE TERMINATED BY A SEMICOLON (;).  ID'S #
        #  MAY BE UP TO 12 CHARACTERS LONG.                #
        #  ----------------------------------------------- #
SELECT ROWS DUTYA (T1-T3) 'SHAKE DOWN TASKS';
       ROWS DUTYB (T4-T5) 'OTHER TASKS';
       COLUMNS MALES (H1=1) 'INCUMBENTS OF THE MALE SEX';
       COLUMNS FEMALES (H1=2)
         'INCUMBENTS OF THE FEMALE SEX';
       COLUMNS OLDERNOTING2 (H2.GT.30 & NOT IN G2)
         'INCUMBENTS OLDER THAN 30 AND NOT IN CLUSTER G2'.
        #  ----------------------------------------------- #
        #  NOW THAT THOSE AREAS OF INTEREST IN THE DATABASE #
        #  HAVE BEEN IDENTIFIED AND LABELED, IT IS POSSIBLE TO #
        #  DIRECT PROCESSING AT THOSE AREAS.               #
        #  THE FOLLOWING DESCRIBE COMMANDS WILL GENERATE A #
        #  TOTAL OF FIVE NEW COLUMNS TO BE STORED ON THE   #
        #  DATABASE.  THE FIRST DESCRIBE STATEMENT (IT IS  #
        #  REALLY THREE DESCRIBE STATEMENTS, BUT SINCE THE #
        #  SAME AREA OF THE DATABASE IS BEING ACCESSED CODING #
        #  CAN BE REDUCED THROUGH THE USE OF THE TERMINATING #
        #  SEMICOLON) IS GENERATING THREE COLUMNS: PERCENT #
        #  PERFORMING PER TASK, AVERAGE PER TASK FOR THOSE #
        #  PERFORMING AND AVERAGE PER TASK FOR THOSE       #
        #  PERFORMING OR NOT.  THE THREE COLUMNS ARE       #
        #  RESPECTIVELY BEING ASSIGNED THE ID'S G5PCNT, G5AVGP #
        #  AND G5AVGA.  THE CALCULATIONS WILL BE PERFORMED #
        #  ACROSS THE COLUMNS ASSOCIATED WITH THE CLUSTER G5 #
        #  (I4-I7).  EACH OF THESE THREE COLUMNS WILL CONTAIN #
        #  5 VALUES (ONE FOR EACH TASK ON THE SAMPLE       #
        #  DATABASE--TASKS, USED IN THE STATEMENT, IS A    #
        #  CODAP80 SYSTEM MODULE ASSOCIATED WITH ALL THE TASKS #
        #  IN THE STUDY).                                  #
        #                                                  #
        #  THE LAST TWO DESCRIBE STATEMENTS ARE CALCULATING #
        #  PERCENT PERFORMING TASKS ACROSS THOSE COLUMNS   #
        #  ASSOCIATED WITH THE CREATED GROUP ID'S MALES (I2, #
        #  I4-I7) AND FEMALES (I1, I3).  THE TWO GENERATED #
        #  COLUMNS ARE ASSIGNED THE ID'S MALESPCNT AND     #
        #  FEMALESPCNT.                                    #
        #                                                  #
        #  ALL FIVE GENERATED COLUMNS WILL BE SAVED ON THE #
        #  PERMANENT DATABASE ALONG WITH THEIR ASSOCIATED  #
        #  REMARKS (HAD THE NOSAVE KEYWORD APPEARED, THE   #
        #  ASSOCIATED COLUMNS WOULD ONLY BE KEPT FOR THE   #
        #  DURATION OF THIS RUN).
```

```
    # THE FOLLOWING DESCRIBE COMMANDS ARE PERFORMING   #
    # THEIR  CALCULATIONS  ON  TASKS  (ROWS)  ACROSS   #
    # INCUMBENTS (COLUMNS). THERE ARE NO RESTRICTIONS ON #
    # WHICH ROWS OF THE DATABASE THE DESCRIBE COMMAND MAY #
    # PROCESS.  THESE VERY SAME CALCULATIONS COULD JUST #
    # AS WELL HAVE BEEN AIMED AT HISTORY INFORMATION, OR #
    # ANY OTHER AGGREGATE OF ROWS SELECTED AND LABELED BY #
    # THE SELECT PROCEDURE.  DESCRIBE MAY ALSO PROCESS  #
    # COLUMNS ACROSS ROWS.  THIS FEATURE GIVES IT THE   #
    # CAPABILITY OF SYMMETRY.                           #
    # ------------------------------------------------- #
DESCRIBE ROWS TASKS FOR (G5)
    G5PCNT := PCNT 'PERCENT PERFORMING TASKS--G5';
    G5AVGP := AVGP
     'AVERAGE PERCENT TIME SPENT (PERFORMING)--G5';
    G5AVGA := AVGA 'AVERAGE PERCENT TIME SPENT (ALL)--G5'.
DESCRIBE ROWS TASKS FOR (MALES)
    MALESPCNT := PCNT 'PERCENT PERFORMING TASKS--MALES'.
DESCRIBE ROWS TASKS FOR (FEMALES)
    FEMALESPCNT := PCNT
     'PERCENT PERFORMING TASKS--FEMALES'.
    # ------------------------------------------------- #
    # A  VALUABLE  STATISTIC  IN  JOB  ANALYSIS  IS  THE #
    # DIFFERENCE IN PERCENT PERFORMING ON TASKS BETWEEN #
    # INCUMBENT AGGREGATES OF INTEREST.  TO CALCULATE   #
    # SUCH A DIFFERENCE STATISTIC, THE USER WOULD EXECUTE #
    # THE  CREATE  PROCEDURE.   IN  THE  ABOVE  DESCRIBE #
    # EXAMPLES, TWO PERCENT PERFORMING COLUMNS WERE     #
    # GENERATED--MALESPCNT & FEMALESPCNT. TO CALCULATE  #
    # THE DIFFERENCE BETWEEN THOSE TWO COLUMNS, BUT ONLY #
    # FOR TASKS 1-3, YOU WOULD EXECUTE THE FOLLOWING    #
    # CREATE COMMAND.                                   #
    # ------------------------------------------------- #
CREATE COLUMN DUTYA DIFFSEX := MALESPCNT-FEMALESPCNT
    'DIFFERENCE IN PERCENT PERFORMING BETWEEN SEXES'.
    # ------------------------------------------------- #
    # THE ABOVE CREATE COMMAND HAS 'CREATED' A NEW      #
    # COLUMN.  THE NEW COLUMN HAS BEEN GIVEN THE ID     #
    # DIFFSEX AND IT, ALONG WITH ITS ASSOCIATED REMARK, #
    # HAS BEEN SAVED ON THE PERMANENT DATABASE. DIFFSEX #
    # WILL HAVE THREE VALUES IN IT, ONE FOR EVERY TASK  #
    # ASSOCIATED WITH THE MODULE ID DUTYA (DUTYA WAS    #
    # FORMED BY AN EARLIER SELECT COMMAND, AND WAS      #
    # ASSIGNED TASKS 1-3).                              #
    #                                                   #
    # THE ABOVE EXAMPLE OF THE CREATE PROCEDURE IS ONE OF #
    # THE SIMPLEST.  CREATE IS A VERY POWERFUL PROCEDURE, #
    # AND ALSO HAS SYMMETRIC CAPABILITY.                #
    # ------------------------------------------------- #
    # ------------------------------------------------- #
    # AT THIS POINT, THE JOB ANALYST MAY WISH TO SEE SOME  #
    # OF THE DATA THAT HAS BEEN GENERATED.  AT PRESENT,    #
    # THE GENERATED DATA IS RESIDING ON THE DATABASE.      #
    # MANY MORE CALCULATIONS COULD BE PERFORMED ON THE     #
    # DATABASE, AND MANY MORE PROCEDURES COULD BE          #
    # EXECUTED.                                            #
    #                                                      #
    # TO PRODUCE REPORTS OF DATA RESIDING ON THE           #
    # DATABASE, THE PRINT PROCEDURE IS EXECUTED.           #
    # THE FIRST PRINT COMMAND WILL PRODUCE A REPORT        #
    # SIMILAR TO THAT PRODUCED BY THE PRTVAR PROGRAM IN    #
    # THE IBM EXPORT VERSION OF CODAP.                     #
    # ---------------------------------------------------- #
PRINT COLUMNS (G6) NOREMARKS / ROWS (HVARS)
    HEADING := 'EXAMPLE 1 OF PRINT'
      'A PRTVAR-LIKE REPORT'.
    # ---------------------------------------------------- #
    # EXAMPLE 1 OF PRINT WILL PRODUCE A REPORT WITH        #
    # INCUMBENTS DOWN THE VERTICAL AXIS AND ALL HISTORY    #
    # INFORMATION ACROSS THE HORIZONTAL AXIS.              #
    #                                                      #
    # THE NEXT EXAMPLE OF PRINT WILL PRODUCE A REPORT      #
    # SIMILAR TO THAT OF THE JOBDEC PROGRAM IN THE IBM     #
    # VERSION OF CODAP.  THE REPORT WILL BE IN TASK        #
    # INVENTORY ORDER.                                     #
    # ---------------------------------------------------- #
PRINT ROWS (TASKS) / COLUMNS (G5PCNT G5AVGP G5AVGA)
    CUM (G5AVGA)
    HEADING := 'EXAMPLE 2 OF PRINT'
      'REPORT SIMILAR TO THAT OF IBM CODAP JOBDEC'
      'AN ACCUMULATION OF G5AVGA HAS BEEN REQUESTED'
      'OUTPUT IS IN TASK INVENTORY ORDER'.
    # ---------------------------------------------------- #
    # THE THIRD EXAMPLE OF PRINT WILL PRODUCE A REPORT     #
    # SIMILAR TO THAT GENERATED ABOVE, EXCEPT THAT IT      #
    # WILL BE BROKEN-DOWN INTO MODULES (DUTYA & DUTYB).    #
    # THE TASKS WITHIN THE MODULES WILL BE SORTED IN       #
    # DESCENDING G5AVGA ORDER.                             #
    # ---------------------------------------------------- #
PRINT ROWS (DUTYA DUTYB) / COLUMNS (G5PCNT G5AVGP
  G5AVGA) SORT DESCENDING BY (G5AVGA)
    HEADING := 'EXAMLPLE 3 OF PRINT'
      'REPORT IS BROKEN-DOWN INTO MODULES'
      'TASKS WITHIN MODULES IN DESCENDING G5AVGA ORDER'.
    # ---------------------------------------------------- #
    # THE LAST (FOURTH) PRINT EXAMPLE WILL PRODUCE A       #
    # GROUP DIFFERENCE DESCRIPTION IN TASK INVENTORY       #
    # ORDER.                                               #
    # ---------------------------------------------------- #
```

```
PRINT ROWS (TASKS) / COLUMNS (MALESPCNT FEMALESPCNT
    DIFFSEX)
    HEADING := 'EXAMPLE 4 OF PRINT'
               'GROUP DIFFERENCE DESCRIPTION'
               'REPORT IS IN TASK INVENTORY ORDER'.
    # ------------------------------------------------------ #
    # THE FOURTH PRINT EXAMPLE WILL GIVE AN IDEA OF HOW     #
    # MISSING VALUES ARE HANDLED IN THE CODAP80 SYSTEM.     #
    # THIS PRINT IS REQUESTING THAT ALL THE TASK VALUES     #
    # OF MALESPCNT, FEMALESPCNT AND DIFFSEX BE PRINTED      #
    # (TASKS DOWN THE VERTICAL AXIS--THE THREE COLUMNS      #
    # ACROSS THE HORIZONTAL AXIS). THERE IS A VALUE OF      #
    # MALESPCNT AND FEMALESPCNT FOR EVERY TASK VALUE, BUT   #
    # DIFFSEX WILL ONLY HAVE VALUES FOR TASKS 1-3           #
    # (DIFFSEX WAS 'CREATED' BY THE CREATE PROCEDURE--BUT   #
    # ONLY FOR THOSE TASKS ASSOCIATED WITH THE MODULE ID    #
    # DUTYA).                                               #
    # ------------------------------------------------------ #
    # ------------------------------------------------------ #
    # THE END STATEMENT MUST TERMINATE ALL CODAP80 SOURCE   #
    # LANGUAGE PROGRAMS.                                    #
    # ------------------------------------------------------ #
END.
```

```
                                                    PAGE - 1
                   STUDY ID - SAMPLEDATA80
                     EXAMPLE 1 OF PRINT
                    A PRTVAR-LIKE REPORT

H - 1   SEX
H - 2   AGE
H - 3   YEARS ON JOB
H - 4   INCUMBENT ID

           H - 1       H - 2       H - 3       H - 4
          -------     -------     -------     -------

G - 6
-------

I - 1       2.00       19.00        1.00        1.00
I - 2       1.00       23.00        2.00        5.00
I - 3       2.00         .         11.00        7.00
I - 4       1.00       41.00       19.00        2.00
I - 5       1.00       27.00        3.00        4.00
I - 6       1.00       53.00       30.00        6.00
I - 7       1.00         .         16.00        3.00
```

```
                                                                 PAGE - 1
                           STUDY ID - SAMPLEDATA80
                            EXAMPLE 2 OF PRINT
                   REPORT SIMILAR TO THAT OF IBM CODAP JOBDEC
                 AN ACCUMULATION OF G5AVGA HAS BEEN REQUESTED
                     OUTPUT IS IN TASK INVENTORY ORDER

G5PCNT PERCENT PERFORMING TASKS--G5
G5AVGP AVERAGE PERCENT TIME SPENT (PERFORMING)--G5
G5AVGA AVERAGE PERCENT TIME SPENT (ALL)--G5
G5AVGA AVERAGE PERCENT TIME SPENT (ALL)--G5

                                                                 ACCUMULATE
                                   G5PCNT    G5AVGP    G5AVGA        G5AVGA
                                   ------    ------    ------    ----------

TASKS
-----

T - 1  SUBDUE VIOLENT INMATES       75.00     23.67     17.75         17.75
T - 2  SHAKE DOWN INMATES          100.00     43.75     43.75         61.50
T - 3  SHAKE DOWN VISITORS          50.00     37.50     18.75         80.25
T - 4  ESCORT INMATES               25.00     22.00      5.50         85.75
T - 5  TESTIFY IN COURT             50.00     28.50     14.25        100.00
```

```
                                                                  PAGE - 1
                              STUDY ID - SAMPLEDATA80
                               EXAMPLE 3 OF PRINT
                         REPORT IS BROKEN-DOWN INTO MODULES
                   TASKS WITHIN MODULES IN DESCENDING G5AVGA ORDER

         G5PCNT  PERCENT PERFORMING TASKS--G5
         G5AVGP  AVERAGE PERCENT TIME SPENT (PERFORMING)--G5
         G5AVGA  AVERAGE PERCENT TIME SPENT (ALL)--G5

                                         G5PCNT      G5AVGP       G5AVGA
                                         ------      ------       ------

         DUTYA   SHAKE DOWN TASKS
         -----

         T - 2   SHAKE DOWN INMATES      100.00       43.75        43.75
         T - 3   SHAKE DOWN VISITORS      50.00       37.50        18.75
         T - 1   SUBDUE VIOLENT INMATES   75.00       23.67        17.75

                 ***************************************************

                                                                  PAGE - 2
                              STUDY ID - SAMPLEDATA80
                               EXAMPLE 3 OF PRINT
                         REPORT IS BROKEN-DOWN INTO MODULES
                   TASKS WITHIN MODULES IN DESCENDING G5AVGA ORDER

         G5PCNT  PERCENT PERFORMING TASKS--G5
         G5AVGP  AVERAGE PERCENT TIME SPENT (PERFORMING)--G5
         G5AVGA  AVERAGE PERCENT TIME SPENT (ALL)--G5

                                         G5PCNT      G5AVGP       G5AVGA
                                         ------      ------       ------

         DUTYB    OTHER TASKS
         -----
         T - 5   TESTIFY IN COURT         50.00       28.50        14.25
         T - 4   ESCORT INMATES           25.00       22.00         5.50
```

```
                                                              PAGE - 1
                         STUDY ID - SAMPLEDATA80
                           EXAMPLE 4 OF PRINT
                       GROUP DIFFERENCE DESCRIPTION
                     REPORT IS IN TASK INVENTORY ORDER

MALESPCNT     PERCENT PERFORMING TASKS--MALES
FEMALESPCNT   PERCENT PERFORMING TASKS--FEMALES
DIFFSEX       DIFFERENCE IN PERCENT PERFORMING BETWEEN SEXES

                                   MALESPCNT   FEMALESPCNT  DIFFSEX
                                   ---------   -----------  -------
TASKS
-----

T - 1    SUBDUE VIOLENT INMATES      80.00         50.00     30.00
T - 2    SHAKE DOWN INMATES         100.00         50.00     50.00
T - 3    SHAKE DOWN VISITORS         60.00        100.00   - 40.00
T - 4    ESCORT INMATES              40.00        100.00       .
T - 5    TESTIFY IN COURT            40.00         50.00       .
```

APPENDIX B

OVERLAP SIMILARITY FORMULAE

OVERLAP SIMILARITY FORMULAE

EUCLIDEAN DISTANCE

$$\text{DISTANCE} = \sum_{i=1}^{i=n} (X_i - Y_i)^2 \; ^{1/2}$$

SQUARED EUCLIDEAN DISTANCE

$$\text{DSQUARE} = \sum_{i=1}^{i=n} (X_i - Y_i)^2$$

ABSOLUTE OVERLAP

$$\text{OVL} = \sum_{i=1}^{i=n} \text{Minimum } (X_i \, , \; Y_i)$$

BINARY

$$\text{BINARY} = \frac{\text{\# Nonzero Elements in Common Between X and Y}}{\text{\# Nonzero X Elements} + \text{\# Nonzero Y Elements} - \text{\# Nonzero Elements in Common Between X and Y}}$$

FORMULAE SYMBOL NOTATION

The symbols X and Y represent the data vectors between which similarity is being calculated. X_i and Y_i represent the ith elements of data vectors X and Y, respectively. The symbol n represents the number of elements in data vectors X or Y.

APPENDIX C

FORTRAN FG PROC
COMPILE, LINK EDIT AND GO *PROCEDURE*
FOR THE G1 FORTRAN COMPILER

FORTRAN FG PROC
COMPILE, LINK EDIT AND GO PROCEDURE FOR THE G1 FORTRAN COMPILER

```
//FG     EXEC  PGM=IEYFORT,REGION=192K
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD SYSOUT=B
//SYSLIN   DD DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//            SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80
/*
//LKED   EXEC  PGM=IEWL,REGION=128K,PARM=(XREF,LET,LIST)
//SYSLIB   DD DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLMOD  DD DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,
//            SPACE=(1024,(20,10,1),RLSE),DCB=BLKSIZE=1024
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSNAME=&SYSUT1,UNIT=SYSDA,SPACE=(1024,(20,10),RLSE),
//            DCB=BLKSIZE=1024
//SYSLIN   DD DSNAME=&LOADSET,DISP=(OLD,DELETE)
//          DD DDNAME=SYSIN
/*
//GO     EXEC PGM=*.LKED.SYSLMOD
//FT05F001 DD DDNAME=SYSIN
//FT06F001 DD SYSOUT=A
//FT07F001 DD SYSOUT=B
```

APPENDIX D

NEW CODAP80 FEATURES

NEW CODAP80 FEATURES

The 83.1 release of CODAP80 includes many new features. Below is a list of changes and additions, including new system features, a new procedure and enhancements to existing programs or procedures.

System Features

Core memory requirements for the CODAP80 interpreter have been reduced to allow its execution in under 820K.

Mass storage requirements for the DECODE file have been reduced by 90%.

A thru operator may now be used to connect created IDs (e.g., CREATEDID1-CREATEDID15).

The assignment operators ":=" and "=" may now be used interchangably.

New Procedure

A new procedure (named RELY) has been added to the CODAP80 interpreter. The procedure calculates inter and intra rater reliabilities on rows or columns of the database.

Database Creation Enhancements

The INPSTD program of the database creation phase of the CODAP80 system allows new ays the input data may be handled. The user has the option of not relativizing task information to a percentage scale. The user also has the option of allowing INPSTD to zero-fill any data that is not right justified. Real numbers may now be read with the format fields specification cards.

The OGROUP program of the database creation phase of the CODAP80 system now allows the user to print the overlap matrix produced during incumbent clustering.

Enhancements to Existing Procedures

PRINT - the PRINT procedure is significantly more efficient, and now provides users with format control over the values that are printed. Two new keywords (NOSKIP and NORESET) have been added to make more efficient use of paper and control how values are accumulated.

VARSUM - if the user specifies both the COUNT and PERCENT keywords of this procedure, execution time is reduced by approximately 50% of that found with using a similar command in the 82.1 release of CODAP80. Column headings are now automatically printed at the top of a new page when an interval needs to be continued. A new keyword (STAT) has been added to provide mean and standard deviation calculations on distribution statistics.

'86

END

DATE

ILMED

8